USER'S GUIDE
3.3

Author

Optimus Prime

About the Author 10

1. Introduction 11

1.1 What is PrimeFaces? 11

1.2 Prime Teknoloji 11

2. Setup 12

2.1 Download 12

2.2 Dependencies 13

2.3 Configuration 13

2.4 Hello World 13

3. Component Suite 14

3.1 AccordionPanel 14

3.2 AjaxBehavior 18

3.3 AjaxStatus 20

3.4 AutoComplete 23

3.5 BlockUI 33

3.6 BreadCrumb 36

3.7 Button 38

3.8 Calendar 41

3.9 Captcha 52

3.10 Carousel 55

3.11 CellEditor 61

3.12 Charts 62

3.12.1 Pie Chart *62*

3.12.2 Line Chart *65*

3.12.3 Bar Chart *68*

3.12.4 Donut Chart *71*

3.12.5 Bubble Chart *74*

3.12.6 Ohlc Chart *77*

3.12.7 MeterGauge Chart *79*

3.12.8 Skinning Charts *81*

3.12.9 Ajax Behavior Events *82*

3.12.10 Charting Tips *83*

3.13 Collector **84**

3.14 Color Picker **85**

3.15 Column **89**

3.16 Columns **91**

3.17 ColumnGroup **92**

3.18 CommandButton **93**

3.19 CommandLink **98**

3.20 ConfirmDialog **101**

3.21 ContextMenu **104**

3.22 Dashboard **107**

3.23 DataExporter **112**

3.24 DataGrid **115**

3.25 DataList **121**

3.26 DataTable **125**

3.27 DefaultCommand **144**

3.28 Dialog **146**

3.29 Drag&Drop **151**

3.29.1 Draggable *151*

3.29.2 Droppable *155*

3.30 Dock 160
3.31 Editor 162
3.32 Effect 166
3.33 FeedReader 169
3.34 Fieldset 170
3.35 FileDownload 174
3.36 FileUpload 177
3.37 Focus 183
3.38 Galleria 185
3.39 GMap 188
3.40 GMapInfoWindow 199
3.41 GraphicImage 200
3.42 Growl 205
3.43 HotKey 209
3.44 IdleMonitor 212
3.45 ImageCompare 214
3.46 ImageCropper 216
3.47 ImageSwitch 220
3.48 Inplace 223
3.49 InputMask 227
3.50 InputText 231
3.51 InputTextarea 234
3.52 Keyboard 238
3.53 Layout 243
3.54 LayoutUnit 248

3.55 LightBox 250

3.56 Log 253

3.57 Media 255

3.58 MegaMenu 257

3.59 Menu 260

3.60 Menubar 265

3.61 MenuButton 268

3.62 MenuItem 270

3.63 Message 273

3.64 Messages 275

3.65 NotificationBar 278

3.66 OrderList 280

3.67 OutputLabel 284

3.68 OutputPanel 287

3.69 OverlayPanel 289

3.70 Panel 292

3.71 PanelGrid 295

3.72 Password 298

3.73 PhotoCam 303

3.74 PickList 305

3.75 Poll 311

3.76 Printer 314

3.77 ProgressBar 315

3.78 Push 319

3.79 RadioButton 320

3.80 Rating 321

3.81 RemoteCommand 325

3.82 Resizable 327

3.83 Ring 331

3.84 Row 334

3.85 RowEditor 335

3.86 RowExpansion 336

3.87 RowToggler 337

3.88 Schedule 338

3.89 ScrollPanel 346

3.90 SelectBooleanButton 348

3.91 SelectBooleanCheckbox 350

3.92 SelectChecboxMenu 352

3.93 SelectManyButton 354

3.94 SelectManyCheckbox 356

3.95 SelectManyMenu 358

3.96 SelectOneButton 360

3.97 SelectOneListbox 362

3.98 SelectOneMenu 364

3.99 SelectOneRadio 368

3.100 Separator 371

3.101 Sheet 373

3.102 SlideMenu 376

3.103 Slider 379

3.104 Spacer 384

3.105 Spinner 385
3.106 SplitButton 390
3.107 Submenu 394
3.108 Stack 395
3.109 SubTable 397
3.110 SummaryRow 398
3.111 Tab 399
3.112 TabView 400
3.113 TagCloud 405
3.114 Terminal 407
3.115 ThemeSwitcher 409
3.116 TieredMenu 411
3.117 Toolbar 414
3.118 ToolbarGroup 416
3.119 Tooltip 417
3.120 Tree 420
3.121 TreeNode 428
3.122 TreeTable 429
3.123 Watermark 432
3.124 Wizard 434
4. Partial Rendering and Processing 440
4.1 Partial Rendering 440
4.1.1 Infrastructure *440*
4.1.2 Using IDs *440*
4.1.3 Notifying Users *442*

4.1.4 Bits&Pieces 443
4.2 Partial Processing 444
4.2.1 Partial Validation 444
4.2.2 Keywords 445
4.2.3 Using Ids 445
4.3 PFS (PrimeFaces Selectors) 446
5. PrimeFaces Mobile 448
6. PrimeFaces Push 449
6.1 Setup 449
6.2 Push API 450
6.3 Push Component 450
6.4 Samples 450
6.4.1 Counter 450
6.4.2 Chat 451
7. Javascript API 453
7.1 PrimeFaces Namespace 453
7.2 Ajax API 454
8. Themes 456
8.1 Applying a Theme 457
8.2 Creating a New Theme 458
8.3 How Themes Work 459
8.4 Theming Tips 460
9. Utilities 461
9.1 RequestContext 461
9.2 EL Functions 464

10. Portlets 466
10.1 Dependencies 466
10.2 Configuration 467
11. Integration with Java EE 470
12. IDE Support 471
12.1 NetBeans 471
12.2 Eclipse 472
13. Project Resources 473
14. FAQ 474

About the Author

Çağatay Çivici (a.k.a Optimus Prime in PrimeFaces Community) is a member of JavaServer Faces Expert Group, the founder and project lead of PrimeFaces and PMC member of open source JSF implementation Apache MyFaces. He's a recognized speaker in international conferences including SpringOne, Jazoon, JAX, W-JAX, JSFSummit, JSFDays, Con-Fess and many local events such as JUGs.

Çağatay is also an author and technical reviewer of a couple books regarding web application development with Java and JSF. As an experienced trainer, he has trained over 300 developers on Java EE technologies mainly JSF, Spring, EJB 3.x and JPA.

Çağatay is currently working as a principal consultant for Prime Teknoloji, the company he co-founded in Turkey.

1. Introduction

1.1 What is PrimeFaces?

PrimeFaces is an open source JSF component suite with various extensions.

- Rich set of components (HtmlEditor, Dialog, AutoComplete, Charts and many more).
- Built-in Ajax based on standard JSF 2.0 Ajax APIs.
- Lightweight, one jar, zero-configuration and no required dependencies.
- Ajax Push support via websockets.
- Mobile UI kit to create mobile web applications for handheld devices.
- Skinning Framework with 30 built-in themes and support for visual theme designer tool.
- Extensive documentation.
- Large, vibrant and active user community.
- Developed with "passion" from application developers to application developers.

1.2 Prime Teknoloji

PrimeFaces is created and maintained by Prime Teknoloji, a Turkish software development company specialized in Agile and Java EE consulting. PrimeFaces Team members are full time engineers at Prime Teknoloji.

- Çağatay Çivici - Architect and Lead Developer
- Yiğit Darçın - Core Developer / QA&Test
- Mustafa Daşgın - Core Developer / QA&Test
- Ömer Gök - Core Developer / QA&Test
- Utku Sönmez - Tester
- Cenk Çivici - Mentor

2. Setup

2.1 Download

PrimeFaces has a single jar called **primefaces-{version}.jar**. There are two ways to download this jar, you can either download from PrimeFaces homepage or if you are a maven user you can define it as a dependency.

Download Manually

Three different artifacts are available for each PrimeFaces version, binary, sources and bundle. Bundle contains binary, sources and javadocs.

http://www.primefaces.org/downloads.html

Download with Maven

Group id of the dependency is *org.primefaces* and artifact id is *primefaces*.

```
<dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>3.3</version>
</dependency>
```

In addition to the configuration above you also need to add PrimeFaces maven repository to the repository list so that maven can download it.

```
<repository>
      <id>prime-repo</id>
      <name>Prime Repo</name>
      <url>http://repository.primefaces.org</url>
</repository>
```

2.2 Dependencies

PrimeFaces only requires a JAVA 5+ runtime and a JSF 2.x implementation as mandatory dependencies. There're some optional libraries for certain features.

Dependency	Version *	Type	Description
JSF runtime	2.0 or 2.1	Required	Apache MyFaces or Oracle Mojarra
itext	2.1.7	Optional	DataExporter (PDF).
apache poi	3.7	Optional	DataExporter (Excel).
rome	1.0	Optional	FeedReader.
commons-fileupload	1.2.1	Optional	FileUpload
commons-io	1.4	Optional	FileUpload

* Listed versions are tested and known to be working with PrimeFaces, other versions of these dependencies may also work but not tested.

2.3 Configuration

PrimeFaces does not require any mandatory configuration.

2.4 Hello World

Once you have added the downloaded jar to your classpath, you need to add the PrimeFaces namespace to your page to begin using the components. Here is a simple page;

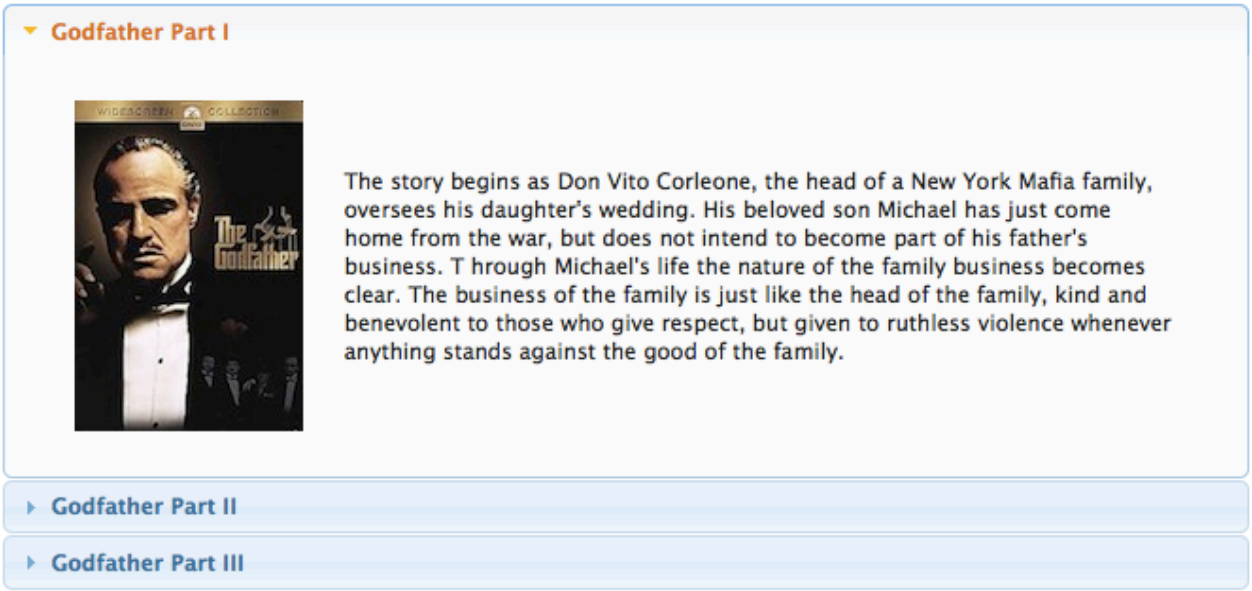
```
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">

<h:head>
</h:head>

<h:body>
      <p:editor />
</h:body>

</html>
```

When you run this page, you should see a rich text editor.

3. Component Suite

3.1 AccordionPanel

AccordionPanel is a container component that displays content in stacked format.


Godfather Part I
The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. T hrough Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.
Godfather Part II
Godfather Part III

Info

Tag	**accordionPanel**
Component Class	**org.primefaces.component.accordionpanel.Accordionpanel**
Component Type	**org.primefaces.component.AccordionPanel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.AccordionPanelRenderer**
Renderer Class	**org.primefaces.component.accordionpanel.AccordionPanelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An EL expression that maps to a server side UIComponent instance in a backing bean.

Name	Default	Type	Description
activeIndex	0	String	Index of the active tab or a comma separated string of indexes when multiple mode is on.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
onTabChange	null	String	Client side callback to invoke when an inactive tab is clicked.
onTabShow	null	String	Client side callback to invoke when a tab gets activated.
dynamic	FALSE	Boolean	Defines the toggle mode.
cache	TRUE	Boolean	Defines if activating a dynamic tab should load the contents from server again.
value	null	java.util.List	List to iterate to display dynamic number of tabs.
var	null	String	Name of iterator to use in a dynamic number of tabs.
multiple	FALSE	Boolean	Controls multiple selection.
widgetVar	null	String	Name of the client side widget.

Getting Started with Accordion Panel

Accordion panel consists of one or more tabs and each tab can group any content. Titles can also be defined with "title" facet.

```
<p:accordionPanel>
      <p:tab title="First Tab Title">
            <h:outputText value= "Lorem"/>
            ...More content for first tab
      </p:tab>
      <p:tab title="Second Tab Title">
            <h:outputText value="Ipsum" />
      </p:tab>
      //any number of tabs
</p:accordionPanel>
```

Dynamic Content Loading

AccordionPanel supports lazy loading of tab content, when dynamic option is set true, only active tab contents will be rendered to the client side and clicking an inactive tab header will do an ajax request to load the tab contents.

This feature is useful to reduce bandwidth and speed up page loading time. By default activating a previously loaded dynamic tab does not initiate a request to load the contents again as tab is cached. To control this behavior use *cache* option.

```
<p:accordionPanel dynamic="true">
      //..tabs
</p:accordionPanel>
```

Client Side Callbacks

onTabChange is called before a tab is shown and *onTabShow* is called after. Both receive container element of the tab to show as the parameter.

```
<p:accordionPanel onTabChange="handleChange(panel)">
      //..tabs
</p:accordionPanel>

<script type="text/javascript">
      function handleChange(panel) {
            //panel: new tab content container
      }
</script>
```

Ajax Behavior Events

tabChange is the one and only ajax behavior event of accordion panel that is executed when a tab is toggled.

```
<p:accordionPanel>
      <p:ajax event="tabChange" listener="#{bean.onChange}" />
</p:accordionPanel>
```

```
public void onChange(TabChangeEvent event) {
      //Tab activeTab = event.getTab();
      //...
 }
```

Your listener(if defined) will be invoked with an *org.primefaces.event.TabChangeEvent* instance that contains a reference to the new active tab and the accordion panel itself.

Dynamic Number of Tabs

When the tabs to display are not static, use the built-in iteration feature similar to ui:repeat.

```
<p:accordionPanel value="#{bean.list}" var="listItem">
      <p:tab title="#{listItem.propertyA}">
            <h:outputText value= "#{listItem.propertyB}"/>
            ...More content
      </p:tab>
</p:accordionPanel>
```

Disabled Tabs

A tab can be disabled by setting disabled attribute to true.

```
<p:accordionPanel>
      <p:tab title="First Tab Title" disabled=”true”>
            <h:outputText value= "Lorem"/>
            ...More content for first tab
      </p:tab>
      <p:tab title="Second Tab Title">
            <h:outputText value="Ipsum" />
      </p:tab>
      //any number of tabs
</p:accordionPanel>
```

Multiple Selection

By default, only one tab at a time can be active, enable *multiple* mode to activate multiple tabs.

```
<p:accordionPanel multiple=”true”>
      //tabs
</p:accordionPanel>
```

Client Side API

Widget: *PrimeFaces.widget.AccordionPanel*

Method	Params	Return Type	Description
select(index)	index: Index of tab to display	void	Activates tab with given index.
unselect(index)	index: Index of tab to hide	void	Deactivates tab with given index.

Skinning

AccordionPanel resides in a main container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Class	Applies
.ui-accordion	Main container element
.ui-accordion-header	Tab header
.ui-accordion-content	Tab content

As skinning style classes are global, see the main Skinning section for more information.

3.2 AjaxBehavior

AjaxBehavior is an extension to standard f:ajax.

Info

Tag	**ajax**
Behavior Id	**org.primefaces.component.AjaxBehavior**
Behavior Class	**org.primefaces.component.behavior.ajax.AjaxBehavior**

Attributes

Name	Default	Type	Description
listener	null	MethodExpr	Method to process in partial request.
immediate	FALSE	boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component(s) to process in partial request.
update	null	String	Component(s) to update with ajax.
onstart	null	String	Callback to execute before ajax request is begins.
oncomplete	null	String	Callback to execute when ajax request is completed.
onsuccess	null	String	Callback to execute when ajax request succeeds.
onerror	null	String	Callback to execute when ajax request fails.
global	TRUE	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus.
disabled	FALSE	Boolean	Disables ajax behavior.
event	null	String	Client side event to trigger ajax request.

Getting Started with AjaxBehavior

AjaxBehavior is attached to the component to ajaxify.

```
 <h:inputText value="#{bean.text}">
       <p:ajax update="out" />
</h:inputText>
<h:outputText id="out" value="#{bean.text}" />
```

In the example above, each time the input changes, an ajax request is sent to the server. When the response is received output text with id "out" is updated with value of the input.

Listener

In case you need to execute a method on a backing bean, define a listener;

```
<h:inputText id="counter">
      <p:ajax update="out" listener="#{counterBean.increment}"/>
</h:inputText>

<h:outputText id="out" value="#{counterBean.count}" />
```

```
public class CounterBean {

      private int count;

      public int getCount() {
            return count;
      }

      public void setCount(int count) {
            this.count = count;
      }

      public void increment() {
            count++;
      }
}
```

Events

Default client side events are defined by components that support client behaviors, for input components it is *onchange* and for command components it is *onclick*. In order to override the dom event to trigger the ajax request use *event* option. In following example, ajax request is triggered when key is up on input field.

```
<h:inputText id="firstname" value="#{bean.text}">
      <p:ajax update="out" event="keyup"/>
</h:inputText>

<h:outputText id="out" value="#{bean.text}" />
```

Partial Processing

Partial processing is used with *process* option which defaults to @this, meaning the ajaxified component. See section 5 for detailed information on partial processing.

3.3 AjaxStatus

AjaxStatus is a global notifier for ajax requests made by PrimeFaces components.

Info

Tag	**ajaxStatus**
Component Class	**org.primefaces.component.ajaxstatus.AjaxStatus**
Component Type	**org.primefaces.component.AjaxStatus**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.AjaxStatusRenderer**
Renderer Class	**org.primefaces.component.ajaxstatus.AjaxStatusRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
onstart	null	String	Client side callback to execute after ajax requests start.
oncomplete	null	String	Client side callback to execute after ajax requests complete.
onprestart	null	String	Client side callback to execute before ajax requests start.
onsuccess	null	String	Client side callback to execute after ajax requests completed succesfully.
onerror	null	String	Client side callback to execute when an ajax request fails.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
widgetVar	null	String	Name of the client side widget.

Getting Started with AjaxStatus

AjaxStatus uses facets to represent the request status. Most common used facets are *start* and *complete*. Start facet will be visible once ajax request begins and stay visible until it's completed. Once the ajax response is received start facet becomes hidden and complete facet shows up.

```
<p:ajaxStatus>
      <f:facet name="start">
            <p:graphicImage value="ajaxloading.gif" />
      </f:facet>

      <f:facet name="complete">
            <h:outputText value="Done!" />
      </f:facet>
</p:ajaxStatus>
```

Events

Here is the full list of available event names;

- *default*: Initially visible when page is loaded.
- *prestart*: Before ajax request is sent.
- *start*: After ajax request begins.
- *success*: When ajax response is received without error.
- *error*: When ajax response is received with error.
- *complete*: When everything finishes.

```
<p:ajaxStatus>
      <f:facet name="prestart">
            <h:outputText value="Starting..." />
      </f:facet>

      <f:facet name="error">
            <h:outputText value="Error" />
      </f:facet>

      <f:facet name="success">
            <h:outputText value="Success" />
      </f:facet>

      <f:facet name="default">
            <h:outputText value="Idle" />
      </f:facet>

      <f:facet name="start">
            <h:outputText value="Sending" />
      </f:facet>

      <f:facet name="complete">
            <h:outputText value="Done" />
      </f:facet>
</p:ajaxStatus>
```

Custom Events

Facets are the declarative way to use, if you'd like to implement advanced cases with scripting you can take advantage of on* callbacks which are the event handler counterparts of the facets.

```
<p:ajaxStatus onstart="alert('Start')" oncomplete="alert('End')"/>
```

A comman usage of programmatic approach is to implement a custom status dialog;

```
<p:ajaxStatus onstart="status.show()" oncomplete="status.hide()"/>

<p:dialog widgetVar="status" modal="true" closable="false">
     Please Wait
</p:dialog>
```

Client Side API

Widget: *PrimeFaces.widget.AjaxStatus*

Method	Params	Return Type	Description
bindFacet(eventName, facetId)	eventName: Name of status event, facetId: Element id of facet container	void	Binds a facet to an event
bindCallback(eventName, fn)	eventName: Name of status event, fn: function to bind	void	Binds a function to an event

Skinning

AjaxStatus is equipped with style and styleClass. Styling directly applies to a container element which contains the facets.

```
<p:ajaxStatus style="width:32px;height:32px" ... />
```

Tips

- Avoid updating ajaxStatus itself to prevent duplicate facet/callback bindings.
- Provide a fixed width/height to the ajaxStatus to prevent page layout from changing.
- Components like commandButton has an attribute *(global)* to control triggering of AjaxStatus.
- There is an ajax loading gif bundled with PrimeFaces which you can use;

3.4 AutoComplete

AutoComplete provides live suggestions while an input is being typed.

Info

Tag	**autoComplete**
Component Class	**org.primefaces.component.autocomplete.AutoComplete**
Component Type	**org.primefaces.component.AutoComplete**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.AutoCompleteRenderer**
Renderer Class	**org.primefaces.component.autocomplete.AutoCompleteRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.

Name	Default	Type	Description
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required.
validator	null	MethodExpr	A method expression that refers to a method validationg the input.
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fails.
widgetVar	null	String	Name of the client side widget.
completeMethod	null	MethodExpr	Method providing suggestions.
var	null	String	Name of the iterator used in pojo based suggestion.
itemLabel	null	String	Label of the item.
itemValue	null	String	Value of the item.
maxResults	unlimited	Integer	Maximum number of results to be displayed.
minQueryLength	1	Integer	Number of characters to be typed before starting to query.
queryDelay	300	Integer	Delay to wait in milliseconds before sending each query to the server.
forceSelection	FALSE	Boolean	When enabled, autoComplete only accepts input from the selection list.
onstart	null	String	Client side callback to execute before ajax request to load suggestions begins.
oncomplete	null	String	Client side callback to execute after ajax request to load suggestions completes.
global	TRUE	Boolean	Defines whether to trigger ajaxStatus or not.
scrollHeight	null	Integer	Defines the height of the items viewport.
effect	null	String	Effect to use when showing/hiding suggestions.
effectDuration	400	Integer	Duration of effect in milliseconds.
dropdown	FALSE	Boolean	Enables dropdown mode when set true.
panelStyle	null	String	Inline style of the items container element.

Name	Default	Type	Description
panelStyleClass	null	String	Style class of the items container element.
multiple	null	Boolean	When true, enables multiple selection.
process	null	String	Component(s) to process on query request.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.

Name	Default	Type	Description
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Getting Started with AutoComplete

Suggestions are loaded by calling a server side completeMethod that takes a single string parameter which is the text entered. Since autoComplete is an input component, it requires a value as usual.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}" />
```

```
public class Bean {

	private String text;

	public List<String> complete(String query) {
		List<String> results = new ArrayList<String>();

		for (int i = 0; i < 10; i++)
			results.add(query + i);

		return results;
	}

	//getter setter
}
```

Pojo Support

Most of the time, instead of simple strings you would need work with your domain objects, autoComplete supports this common use case with the use of a converter and data iterator.

Following example loads a list of players, itemLabel is the label displayed as a suggestion and itemValue is the submitted value. Note that when working with pojos, you need to plug-in your own converter.

```
<p:autoComplete value="#{playerBean.selectedPlayer}"
      completeMethod="#{playerBean.completePlayer}"
      var="player"
      itemLabel="#{player.name}"
      itemValue="#{player}"
      converter="playerConverter"/>
```

```
public class PlayerBean {

      private Player selectedPlayer;

      public Player getSelectedPlayer() {
            return selectedPlayer;
      }
      public void setSelectedPlayer(Player selectedPlayer) {
            this.selectedPlayer = selectedPlayer;
      }

      public List<Player> complete(String query) {
            List<Player> players = readPlayersFromDatasource(query);

            return players;
      }
}
```

```
public class Player {

      private String name;

      //getter setter
}
```

Limiting the Results

Number of results shown can be limited, by default there is no limit.

```
<p:autoComplete value="#{bean.text}"
                  completeMethod="#{bean.complete}"
                  maxResults="5" />
```

Minimum Query Length

By default queries are sent to the server and completeMethod is called as soon as users starts typing at the input text. This behavior is tuned using the *minQueryLength* attribute.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
                        minQueryLength="3" />
```

With this setting, suggestions will start when user types the 3rd character at the input field.

Query Delay

AutoComplete is optimized using *queryDelay* option, by default autoComplete waits for 300 milliseconds to query a suggestion request, if you'd like to tune the load balance, give a longer value. Following autoComplete waits for 1 second after user types an input.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
                        queryDelay="1000" />
```

Custom Content

AutoComplete can display custom content by nesting columns.

```
<p:autoComplete value="#{autoCompleteBean.selectedPlayer}"
      completeMethod="#{autoCompleteBean.completePlayer}"
      var="p" itemValue="#{p}" converter="player">

      <p:column>
         <p:graphicImage value="/images/barca/#{p.photo}" width="40" height="50"/>
      </p:column>

      <p:column>
            #{p.name} - #{p.number}
      </p:column>
</p:autoComplete>
```

Dropdown Mode

When dropdown mode is enabled, a dropdown button is displayed next to the input field, clicking this button will do a search with an empty query, a regular completeMethod implementation should load all available items as a response.

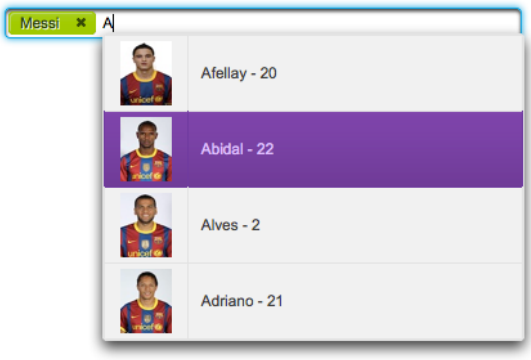
```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
                  dropdown="true" />
```

Multiple Selection

AutoComplete supports multiple selection as well, to use this feature set multiple option to true and define a list as your backend model. Following example demonstrates multiple selection with custom content support.

```
<p:autoComplete id="advanced" value="#{autoCompleteBean.selectedPlayers}"
                  completeMethod="#{autoCompleteBean.completePlayer}"
                   var="p" itemLabel="#{p.name}" itemValue="#{p}" converter="player"
                  multiple="true">

      <p:column style="width:20%;text-align:center">
            <p:graphicImage value="/images/barca/#{p.photo}"/>
      </p:column>

      <p:column style="width:80%">
            #{p.name} - #{p.number}
      </p:column>
</p:autoComplete>
```

```
public class AutoCompleteBean {

      private List<Player> selectedPlayers;

      //...
}
```



Ajax Behavior Events

Instead of waiting for user to submit the form manually to process the selected item, you can enable instant ajax selection by using the *itemSelect* ajax behavior. Example below demonstrates how to display a message about the selected item instantly.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}">
      <p:ajax event="itemSelect" listener="bean.handleSelect" update="msg" />
</p:autoComplete>

<p:messages id=”msg” />
```

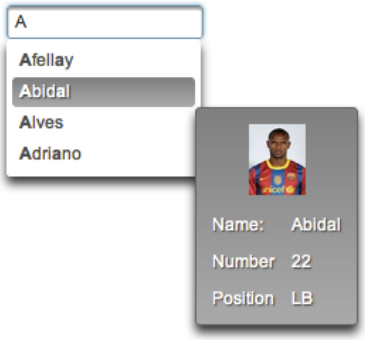
```
public class Bean {

      public void handleSelect(SelectEvent event) {
            Object item = event.getObject();
            FacesMessage msg = new FacesMessage("Selected", "Item:" + item);
      }

      ...
}
```

Your listener(if defined) will be invoked with an *org.primefaces.event.Select* instance that contains a reference to the selected item. Note that autoComplete also supports events inherited from regular input text such as blur, focus, mouseover in addition to *itemSelect*. Similarly, *itemUnselect* event is provided for multiple autocomplete when an item is removed by clicking the remove icon. In this case *org.primefaces.event.Unselect* instance is passed to a listener if defined.

ItemTip

Itemtip is an advanced built-in tooltip when mouse is over on suggested items. Content of the tooltip is defined via the *itemtip* facet.

```
<p:autoComplete value="#{autoCompleteBean.selectedPlayer1}" id="basicPojo"
      completeMethod="#{autoCompleteBean.completePlayer}"
      var="p" itemLabel="#{p.name}" itemValue="#{p}" converter="player">
      <f:facet name="itemtip">
            <h:panelGrid  columns="2" cellpadding="5">
                  <f:facet name="header">
                        <p:graphicImage value="/images/barca/#{p.photo}" />
                  </f:facet>

                  <h:outputText value="Name: " />
                  <h:outputText id="modelNo" value="#{p.name}" />

                  <h:outputText value="Number " />
                  <h:outputText id="year" value="#{p.number}" />

                  <h:outputText value="Position " />
                  <h:outputText value="#{p.position}"/>
            </h:panelGrid>
      </f:facet>
</p:autoComplete>
```



Client Side Callbacks

onstart and *oncomplete* are used to execute custom javascript before and after an ajax request to load suggestions.

```
<p:autoComplete value="#{bean.text}" completeMethod="#{bean.complete}"
     onstart="handleStart(request)" oncomplete="handleComplete(response)" />
```

onstart callback gets a *request* parameter and oncomplete gets a *response* parameter, these parameters contain useful information. For example *request* is the query string and *response* is the xhr request sent under the hood.

Client Side API

Widget: *PrimeFaces.widget.AutoComplete*

Method	Params	Return Type	Description
search(value)	value: keyword for search	void	Initiates a search with given value
close()	-	void	Hides suggested items menu
disable()	-	void	Disables the input field
enable()	-	void	Enables the input field
deactivate()	-	void	Deactivates search behavior
activate()	-	void	Activates search behavior

Skinning

Following is the list of structural style classes;

Class	Applies
.ui-autocomplete	Container element.
.ui-autocomplete-input	Input field.
.ui-autocomplete-panel	Container of suggestions list.
.ui-autocomplete-items	List of items
.ui-autocomplete-item	Each item in the list.
.ui-autocomplete-query	Highlighted part in suggestions.

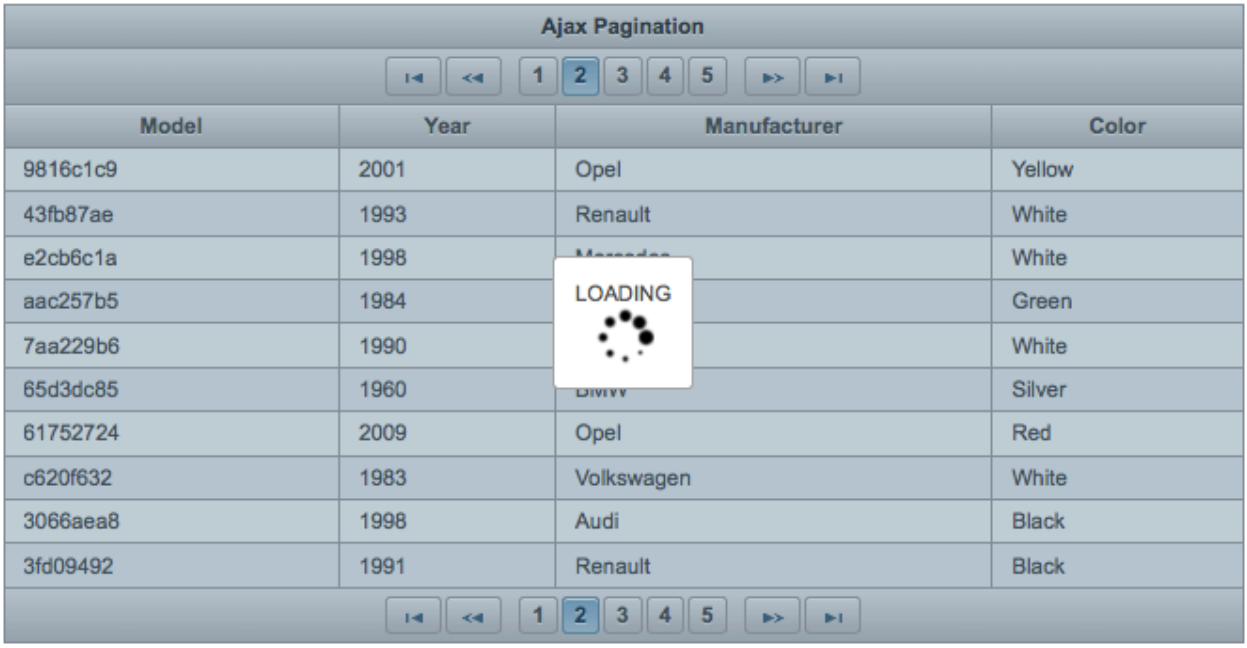
As skinning style classes are global, see the main Skinning section for more information.

Tips

- Do not forget to use a converter when working with pojos.
- Enable forceSelection if you'd like to accept values only from suggested list.
- Increase query delay to avoid unnecessary load to server as a result of user typing fast.

3.5 BlockUI

BlockUI is used to block JSF components during ajax processing.



Info

Tag	**blockUI**
Component Class	**org.primefaces.component.blockui.BlockUI**
Component Type	**org.primefaces.component.BlockUI**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.BlockUIRenderer**
Renderer Class	**org.primefaces.component.blockui.BlockUIRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.

Name	Default	Type	Description
trigger	null	String	Identifier of the component(s) to bind.
block	null	String	Identifier of the component to block.

Getting Started with BlockUI

BlockUI requires *trigger* and *block* attributes to be defined. With the special ajax integration, ajax requests whose source is trigger components will block the ui onstart and unblock oncomplete. Example below blocks the ui of the panel when saveBtn is clicked and unblock when ajax response is received.

```
<p:panel id="pnl" header="My Panel">
      //content

      <p:commandButton id="saveBtn" value="Save" />
</p:panel>

<p:blockUI block="pnl" trigger="saveBtn" />
```

Multiple triggers are defined as a comma separated list.

```
<p:blockUI block="pnl" trigger="saveBtn,deleteBtn,updateBtn" />
```

Custom Content

In order to display custom content like a loading text and animation, place the content inside the blockUI.

```
<p:dataTable id="dataTable" var="car" value="#{tableBean.cars}"
                            paginator="true" rows="10">
      <p:column>
            <f:facet name="header">
                  <h:outputText value="Model" />
            </f:facet>
            <h:outputText value="#{car.model}" />
      </p:column>

      //more columns
</p:dataTable>

<p:blockUI block="dataTable" trigger="dataTable">
      LOADING<br />
      <p:graphicImage value="/images/ajax-loader.gif"/>
</p:blockUI>
```

Client Side API

Widget: *PrimeFaces.widget.BlockUI*

Method	Params	Return Type	Description
show()	-	void	Blocks the UI.
hide()	-	void	Unblocks the UI

Skinning

Following is the list of structural style classes;

Class	Applies
.ui-blockui	Container element.
.ui-blockui-content	Container for custom content.

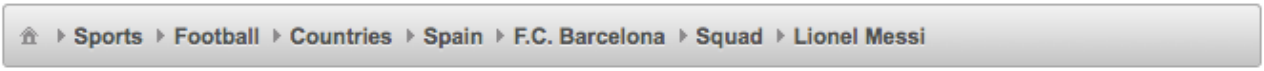
As skinning style classes are global, see the main Skinning section for more information.

Tips

- BlockUI does not support absolute and fixed positioned components at the moment. e.g. dialog.

3.6 BreadCrumb

Breadcrumb is a navigation component that provides contextual information about page hierarchy in the workflow.



Info

Tag	**breadCrumb**
Component Class	**org.primefaces.component.breadcrumb.BreadCrumb**
Component Type	**org.primefaces.component.BreadCrumb**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.BreadCrumbRenderer**
Renderer Class	**org.primefaces.component.breadcrumb.BreadCrumbRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
model	null	MenuModel	MenuModel instance to create menus programmatically
style	null	String	Style of main container element.
styleClass	null	String	Style class of main container

Getting Started with BreadCrumb

Steps are defined as child menuitem components in breadcrumb.

```
<p:breadCrumb>
      <p:menuitem label="Categories" url="#" />
      <p:menuitem label="Sports" url="#" />
      //more menuitems
</p:breadCrumb>
```

Dynamic Menus

Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

Breadcrumb resides in a container element that *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-breadcrumb	Main breadcrumb container element.
.ui-breadcrumb .ui-menu-item-link	Each menuitem.
.ui-breadcrumb .ui-menu-item-text	Each menuitem label.
.ui-breadcrumb-chevron	Seperator of menuitems.

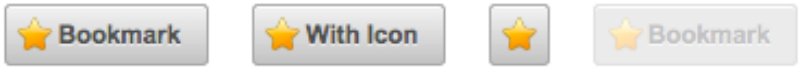
As skinning style classes are global, see the main Skinning section for more information.

Tips

- If there is a dynamic flow, use model option instead of creating declarative p:menuitem components and bind your MenuModel representing the state of the flow.
- Breadcrumb can do ajax/non-ajax action requests as well since p:menuitem has this option. In this case, breadcrumb must be nested in a form.
- url option is the key for a menuitem, if it is defined, it will work as a simple link. If you'd like to use menuitem to execute command with or without ajax, do not define the url option.

3.7 Button

Button is an extension to the standard h:button component with skinning capabilities.



Info

Tag	**button**
Component Class	**org.primefaces.component.button.Button**
Component Type	**org.primefaces.component.Button**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ButtonRenderer**
Renderer Class	**org.primefaces.component.button.ButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
outcome	null	String	Used to resolve a navigation case.
includeViewParams	FALSE	Boolean	Whether to include page parameters in target URI
fragment	null	String	Identifier of the target page which should be scrolled to.
disabled	FALSE	Boolean	Disables button.
accesskey	null	String	Access key that when pressed transfers focus to button.
alt	null	String	Alternate textual description.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
image	null	String	Style class for the button icon. (deprecated: use icon)

Name	Default	Type	Description
lang	null	String	Code describing the language used in the generated markup for this component.
onblur	null	String	Client side callback to execute when button loses focus.
onchange	null	String	Client side callback to execute when button loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when button is clicked.
ondblclick	null	String	Client side callback to execute when button is double clicked.
onfocus	null	String	Client side callback to execute when button receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over button.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over button.
onkeyup	null	String	Client side callback to execute when a key is released over button.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over button.
onmousemove	null	String	Client side callback to execute when a pointer button is moved within button
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from button.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto button.
onmouseup	null	String	Client side callback to execute when a pointer button is released over button.
style	null	String	Inline style of the button.
styleClass	null	String	Style class of the button.
readOnly	FALSE	Boolean	Makes button read only.
tabindex	null	Integer	Position in the tabbing order.
title	null	String	Advisory tooltip informaton.
href	null	String	Resource to link directly to implement anchor behavior.
icon	null	String	Icon of the button.
iconPos	left	String	Position of the button icon.
escape	TRUE	Boolean	Defines whether label would be escaped or not.

Getting Started with Button

p:button usage is same as standard h:button, an outcome is necessary to navigate using GET requests. Assume you are at source.xhtml and need to navigate target.xhtml.

```
<p:button outcome="target" value="Navigate"/>
```

Parameters

Parameters in URI are defined with nested <f:param /> tags.

```
<p:button outcome="target" value="Navigate">
     <f:param name="id" value="10" />
</p:button>
```

Icons

Icons for button are defined via css and *icon* attribute, if you use title instead of value, only icon will be displayed and title text will be displayed as tooltip on mouseover. You can also use icons from PrimeFaces themes.

```
<p:button outcome="target" icon="star" value="With Icon"/>
<p:button outcome="target" icon="star" title="With Icon"/>
```

```
.star {
     background-image: url("images/star.png");
}
```

Skinning

Button renders a *button* tag which *style* and *styleClass* applies. Following is the list of structural style classes;

Style Class	Applies
.ui-button	Button element
.ui-button-text-only	Button element when icon is not used
.ui-button-text	Label of button

As skinning style classes are global, see the main Skinning section for more information.

3.8 Calendar

Calendar is an input component used to select a date featuring display modes, paging, localization, ajax selection and more.

Info

Tag	**calendar**
Component Class	**org.primefaces.component.calendar.Calendar**
Component Type	**org.primefaces.component.Calendar**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CalendarRenderer**
Renderer Class	**org.primefaces.component.calendar.CalendarRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	java.util.Date	Value of the component
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.

Name	Default	Type	Description
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fails.
widgetVar	null	String	Name of the client side widget.
mindate	null	Date or String	Sets calendar's minimum visible date
maxdate	null	Date or String	Sets calendar's maximum visible date
pages	int	1	Enables multiple page rendering.
disabled	FALSE	Boolean	Disables the calendar when set to true.
mode	popup	String	Defines how the calendar will be displayed.
pattern	MM/dd/yyyy	String	DateFormat pattern for localization
locale	null	Locale/String	Locale to be used for labels and conversion.
popupIcon	null	String	Icon of the popup button
popupIconOnly	FALSE	Boolean	When enabled, popup icon is rendered without the button.
navigator	FALSE	Boolean	Enables month/year navigator
timeZone	null	TimeZone	String or a java.util.TimeZone instance to specify the timezone used for date conversion, defaults to TimeZone.getDefault ()
readonlyInput	FALSE	Boolean	Makes input text of a popup calendar readonly.
showButtonPanel	FALSE	Boolean	Visibility of button panel containing today and done buttons.
effect	null	String	Effect to use when displaying and showing the popup calendar.
effectDuration	normal	String	Duration of the effect.
showOn	both	String	Client side event that displays the popup calendar.

Name	Default	Type	Description
showWeek	FALSE	Boolean	Displays the week number next to each week.
disabledWeekends	FALSE	Boolean	Disables weekend columns.
showOtherMonths	FALSE	Boolean	Displays days belonging to other months.
selectOtherMonths	FALSE	Boolean	Enables selection of days belonging to other months.
yearRange	null	String	Year range for the navigator, default "c-10:c +10"
timeOnly	FALSE	Boolean	Shows only timepicker without date.
stepHour	1	Integer	Hour steps.
stepMinute	1	Integer	Minute steps.
stepSecond	1	Integer	Second steps.
minHour	0	Integer	Minimum boundary for hour selection.
maxHour	23	Integer	Maximum boundary for hour selection.
minMinute	0	Integer	Minimum boundary for minute selection.
maxMinute	59	Integer	Maximum boundary for hour selection.
minSecond	0	Integer	Minimum boundary for second selection.
maxSecond	59	Integer	Maximum boundary for second selection.
pagedate	null	Object	Initial date to display if value is null.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.

Name	Default	Type	Description
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.
beforeShowDay	null	String	Client side callback to execute before displaying a date, used to customize date display.

Getting Started with Calendar

Value of the calendar should be a java.util.Date.

```
<p:calendar value="#{dateBean.date}"/>
```

```
public class DateBean {

      private Date date;

      //Getter and Setter
}
```

Display Modes

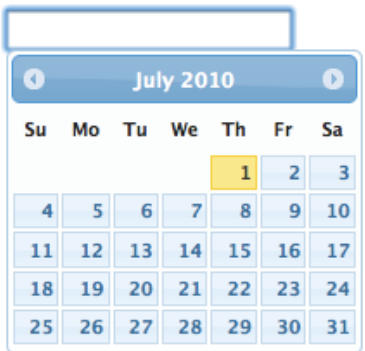
Calendar has two main display modes, *popup* (default) and *inline*.

Inline

```
<p:calendar value="#{dateBean.date}" mode="inline" />
```

Popup

```
<p:calendar value="#{dateBean.date}" mode="popup" />
```

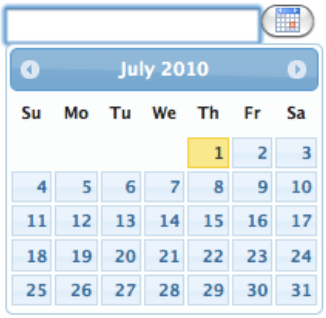


showOn option defines the client side event to display the calendar. Valid values are;

- focus: When input field receives focus
- button: When popup button is clicked
- both: Both *focus* and *button* cases

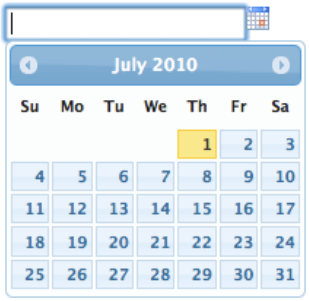
Popup Button

```
<p:calendar value="#{dateBean.date}" mode="popup" showOn="button" />
```



Popup Icon Only

```
<p:calendar value="#{dateBean.date}" mode="popup"
                showOn="button" popupIconOnly="true" />
```



Paging

Calendar can also be rendered in multiple pages where each page corresponds to one month. This feature is tuned with the *pages* attribute.

```
<p:calendar value="#{dateController.date}" pages="3"/>
```

Localization

By default locale information is retrieved from the view's locale and can be overridden by the locale attribute. Locale attribute can take a locale key as a String or a java.util.Locale instance. Default language of labels are English and you need to add the necessary translations to your page manually as PrimeFaces does not include language translations. PrimeFaces Wiki Page for PrimeFacesLocales is a community driven page where you may find the translations you need. Please contribute to this wiki with your own translations.

http://wiki.primefaces.org/display/Components/PrimeFaces+Locales

Translation is a simple javascript object, we suggest adding the code to a javascript file and include in your application. Following is a Turkish calendar.

```
<h:outputScript name="path_to_your_translations.js" />

<p:calendar value="#{dateController.date}" locale="tr" navigator="true"
                                            showButtonPanel="true"/>
```

To override calculated pattern from locale, use the pattern option;

```
<p:calendar value="#{dateController.date1}" pattern="dd.MM.yyyy"/>
<p:calendar value="#{dateController.date2}" pattern="yy, M, d"/>
<p:calendar value="#{dateController.date3}" pattern="EEE, dd MMM, yyyy"/>
```

Effects

Various effects can be used when showing and hiding the popup calendar, options are;

- show
- slideDown
- fadeIn
- blind
- bounce
- clip
- drop
- fold
- slide

Ajax Behavior Events

Calendar provides a *dateSelect* ajax behavior event to execute an instant ajax selection whenever a date is selected. If you define a method as a listener, it will be invoked by passing an *org.primefaces.event.DateSelectEvent* instance.

```
<p:calendar value="#{calendarBean.date}">
      <p:ajax event="dateSelect" listener="#{bean.handleDateSelect}" update="msg" />
</p:calendar>

<p:messages id="msg" />
```

```
public void handleDateSelect(DateSelectEvent event) {
      Date date = event.getDate();
      //Add facesmessage
}
```

In popup mode, calendar also supports regular ajax behavior events like blur, keyup and more.

Date Ranges

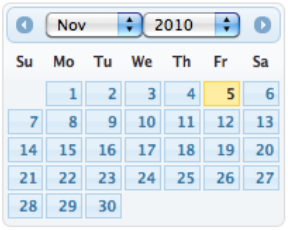
Using mindate and maxdate options, selectable dates can be restricted. Values for these attributes can either be a string or a java.util.Date.

```
<p:calendar value="#{dateBean.date}" mode="inline"
       mindate="07/10/2010" maxdate="07/15/2010"/>
```

Navigator

Navigator is an easy way to jump between months/years quickly.

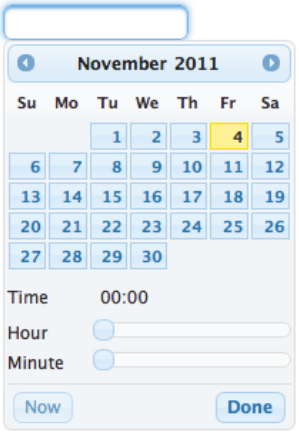
```
<p:calendar value="#{dateBean.date}" mode="inline" navigator="true" />
```



TimePicker

TimePicker functionality is enabled by adding time format to your pattern.

```
<p:calendar value="#{dateBean.date}" pattern="MM/dd/yyyy HH:mm" />
```



Advanced Customization

Use beforeShowDay javascript callback to customize the look of each date. The function returns an array with two values, first one is flag to indicate if date would be displayed as enabled and second parameter is the optional style class to add to date cell. Following example disabled tuesdays and fridays.

```
<p:calendar value="#{dateBean.date}" beforeShowDay="tuesdaysAndFridaysOnly" />
```

```
function tuesdaysAndFridaysDisabled(date) {
      var day = date.getDay();

      return [(day != 2 && day != 5), '']
}
```

Client Side API

Widget: *PrimeFaces.widget.Calendar*

Method	Params	Return Type	Description
getDate()	-	Date	Return selected date
setDate(date)	date: Date to display	void	Sets display date
disable()	-	void	Disables calendar
enable()	-	void	Enables calendar

Skinning

Calendar resides in a container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-datepicker	Main container
.ui-datepicker-header	Header container
.ui-datepicker-prev	Previous month navigator
.ui-datepicker-next	Next month navigator
.ui-datepicker-title	Title
.ui-datepicker-month	Month display
.ui-datepicker-table	Date table
.ui-datepicker-week-end	Label of weekends
.ui-datepicker-other-month	Dates belonging to other months
.ui-datepicker td	Each cell date
.ui-datepicker-buttonpane	Button panel

Style Class	Applies
.ui-datepicker-current	Today button
.ui-datepicker-close	Close button

As skinning style classes are global, see the main Skinning section for more information.

3.9 Captcha

Captcha is a form validation component based on Recaptcha API.

Info

Tag	**captcha**
Component Class	**org.primefaces.component.captcha.Captcha**
Component Type	**org.primefaces.component.Captcha**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CaptchaRenderer**
Renderer Class	**org.primefaces.component.captcha.CaptchaRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required.

Name	Default	Type	Description
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input.
valueChangeListener	null	ValueChange Listener	A method binding expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
publicKey	null	String	Public recaptcha key for a specific domain (deprecated)
theme	red	String	Theme of the captcha.
language	en	String	Key of the supported languages.
tabindex	null	Integer	Position of the input element in the tabbing order.
label	null	String	User presentable field name.
secure	FALSE	Boolean	Enables https support

Getting Started with Captcha

Catpcha is implemented as an input component with a built-in validator that is integrated with reCaptcha. First thing to do is to sign up to reCaptcha to get public&private keys. Once you have the keys for your domain, add them to web.xml as follows;

```
<context-param>
    <param-name>primefaces.PRIVATE_CAPTCHA_KEY</param-name>
    <param-value>YOUR_PRIVATE_KEY</param-value>
</context-param>

<context-param>
    <param-name>primefaces.PUBLIC_CAPTCHA_KEY</param-name>
    <param-value>YOUR_PUBLIC_KEY</param-value>
</context-param>
```

That is it, now you can use captcha as follows;

```
<p:captcha />
```

Themes

Captcha features following built-in themes for look and feel customization.

- red (default)
- white
- blackglass
- clean

Themes are applied via the theme attribute.

```
<p:captcha theme="white"/>
```

Languages

Text instructions displayed on captcha is customized with the *language* attribute. Below is a captcha with Turkish text.

```
<p:captcha language="tr"/>
```

Overriding Validation Messages

By default captcha displays it's own validation messages, this can be easily overridden by the JSF message bundle mechanism. Corresponding keys are;

Summary	primefaces.captcha.INVALID
Detail	primefaces.captcha.INVALID_detail

Tips

- Use label option to provide readable error messages in case validation fails.
- Enable *secure* option to support https otherwise browsers will give warnings.
- See http://www.google.com/recaptcha/learnmore to learn more about how reCaptcha works.

3.10 Carousel

Carousel is a multi purpose component to display a set of data or general content with slide effects.

Info

Tag	**carousel**
Component Class	**org.primefaces.component.carousel.Carousel**
Component Type	**org.primefaces.component.Carousel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CarouselRenderer**
Renderer Class	**org.primefaces.component.carousel.CarouselRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	A value expression that refers to a collection
var	null	String	Name of the request scoped iterator
rows	3	Integer	Number of visible items per page
first	0	Integer	Index of the first element to be displayed
widgetVar	null	String	Name of the client side widget.
circular	FALSE	Boolean	Sets continuous scrolling
vertical	FALSE	Boolean	Sets vertical scrolling

Name	Default	Type	Description
autoPlayInterval	0	Integer	Sets the time in milliseconds to have Carousel start scrolling automatically after being initialized
pageLinks	3	Integer	Defines the number of page links of paginator.
effect	slide	String	Name of the animation, could be "fade" or "slide".
easing	easeInOutCirc	String	Name of the easing animation.
effectDuration	500	Integer	Duration of the animation in milliseconds.
dropdownTemplate.	{page}	String	Template string for dropdown of paginator.
style	null	String	Inline style of the component..
styleClass	null	String	Style class of the component..
itemStyle	null	String	Inline style of each item.
itemStyleClass	null	String	Style class of each item.
headerText	null	String	Label for header.
footerText	null	String	Label for footer.

Getting Started with Carousel

Calendar has two main use-cases; data and general content display. To begin with data iteration let's use a list of cars to display with carousel.

```
public class Car {

      private String model;
      private int year;
      private String manufacturer;
      private String color;
      ...
}
```

```
public class CarBean {

      private List<Car> cars;

      public CarListController() {
            cars = new ArrayList<Car>();
            cars.add(new Car("myModel", 2005, "ManufacturerX", "blue"));
            //add more cars
      }

      //getter setter
}
```

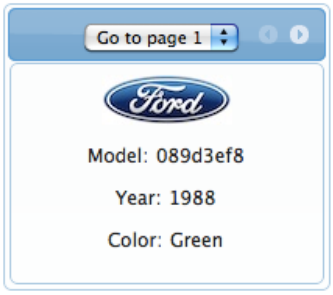
```
<p:carousel value="#{carBean.cars}" var="car" itemStyle="width:200px">
     <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg"/>
     <h:outputText value="Model: #{car.model}" />
     <h:outputText value="Year: #{car.year}" />
     <h:outputText value="Color: #{car.color}" />
</p:carousel>
```

Carousel iterates through the cars collection and renders it's children for each car, note that you also need to define a width for each item.

Limiting Visible Items

Bu default carousel lists it's items in pages with size 3. This is customizable with the rows attribute.

```
<p:carousel value="#{carBean.cars}" var="car" rows="1" itemStyle="width:200px" >
     ...
</p:carousel>
```



Effects

Paging happens with a slider effect by default and following easing options are supported.

- backBoth
- backIn
- backOut
- bounceBoth
- bounceIn
- bounceOut
- easeBoth
- easeBothStrong
- easeIn
- easeInStrong
- easeNone
- easeOut
- easeOutStrong
- elasticBoth
- elasticIn
- elasticOut

Note: Easing names are case sensitive and incorrect usage may result in javascript errors

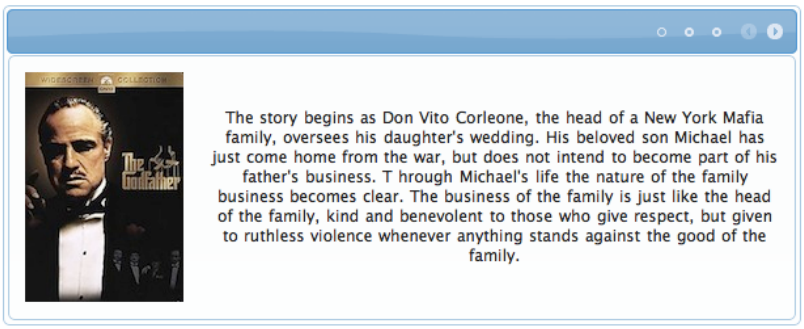
SlideShow

Carousel can display the contents in a slideshow, for this purpose *autoPlayInterval* and *circular* attributes are used. Following carousel displays a collection of images as a slideshow.

```
<p:carousel autoPlayInterval="2000" rows="1" effect="easeInStrong" circular="true"
      itemStyle=”width:200px” >
      <p:graphicImage value="/images/nature1.jpg"/>
      <p:graphicImage value="/images/nature2.jpg"/>
      <p:graphicImage value="/images/nature3.jpg"/>
      <p:graphicImage value="/images/nature4.jpg"/>
</p:carousel>
```

Content Display

Another use case of carousel is tab based content display.


The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. T hrough Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.

```
<p:carousel rows="1" itemStyle="height:200px;width:600px;">
      <p:tab title="Godfather Part I">
            <h:panelGrid columns="2" cellpadding="10">
                  <p:graphicImage value="/images/godfather/godfather1.jpg" />
                  <h:outputText value="The story begins as Don Vito ..." />
            </h:panelGrid>
      </p:tab>
      <p:tab title="Godfather Part II">
            <h:panelGrid columns="2" cellpadding="10">
                  <p:graphicImage value="/images/godfather/godfather2.jpg" />
                  <h:outputText value="Francis Ford Coppola's ..."/>
            </h:panelGrid>
      </p:tab>
      <p:tab title="Godfather Part III">
            <h:panelGrid columns="2" cellpadding="10">
                  <p:graphicImage value="/images/godfather/godfather3.jpg" />
                  <h:outputText value="After a break of ..." />
            </h:panelGrid>
      </p:tab>
</p:carousel>
```

Item Selection

When selecting an item from a carousel with a command component, p:column is necessary to process selection properly. Following example displays selected car contents within a dialog;

```
<h:form id=”form">
 <p:carousel value="#{carBean.cars}" var="car" itemStyle=”width:200px” >
  <p:column>
    <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg"/>
      <p:commandLink update=":form:detail" oncomplete="dlg.show()">
         <h:outputText value="Model: #{car.model}" />
         <f:setPropertyActionListener value="#{car}" target="#{carBean.selected}" />
      </p:commandLink>
  </p:column>
 </p:carousel>

 <p:dialog widgetVar="dlg">
     <h:outputText id="detail" value="#{carBean.selected}" />
 </p:dialog>
</h:form>
```

```
public class CarBean {

      private List<Car> cars;

      private Car selected;

      //getters and setters
}
```

Header and Footer

Header and Footer of carousel can be defined in two ways either, using *headerText* and *footerText* options that take simple strings as labels or by *header* and *footer* facets that can take any custom content.

Skinning

Carousel resides in a container element which *style* and *styleClass* options apply. *itemStyle* and *itemStyleClass* attributes apply to each item displayed by carousel. Following is the list of structural style classes;

Style Class	Applies
.ui-carousel	Main container
.ui-carousel-header	Header container
.ui-carousel-header-title	Header content

Style Class	Applies
.ui-carousel-viewport	Content container
.ui-carousel-button	Navigation buttons
.ui-carousel-next-button	Next navigation button of paginator
.ui-carousel-prev-button	Prev navigation button of paginator
.ui-carousel-page-links	Page links of paginator.
.ui-carousel-page-link	Each page link of paginator.
.ui-carousel-item	Each item.

As skinning style classes are global, see the main Skinning section for more information.

Tips

- Carousel is a NamingContainer, make sure you reference components outside of carousel properly following conventions.

3.11 CellEditor

CellEditor is a helper component of datatable used for incell editing.

Info

Tag	**cellEditor**
Component Class	**org.primefaces.component.celleditor.CellEditor**
Component Type	**org.primefaces.component.CellEditor**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CellEditorRenderer**
Renderer Class	**org.primefaces.component.celleditor.CellEditorRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting Started with CellEditor

See inline editing section in datatable documentation for more information about usage.

3.12 Charts

Charts are used to display graphical data. There're various chart types like pie, bar, line and more.

3.12.1 Pie Chart

Pie chart displays category-data pairs in a pie graphic.

Info

Tag	**pieChart**
Component Class	**org.primefaces.component.chart.pie.PieChart**
Component Type	**org.primefaces.component.chart.PieChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.PieChartRenderer**
Renderer Class	**org.primefaces.component.chart.pie.PieChartRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
seriesColors	null	String	Comma separated list of colors in hex format.
diameter	null	Integer	Diameter of the pie, auto computed by default.
sliceMargin	0	Integer	Gap between slices.
fill	TRUE	Boolean	Render solid slices.

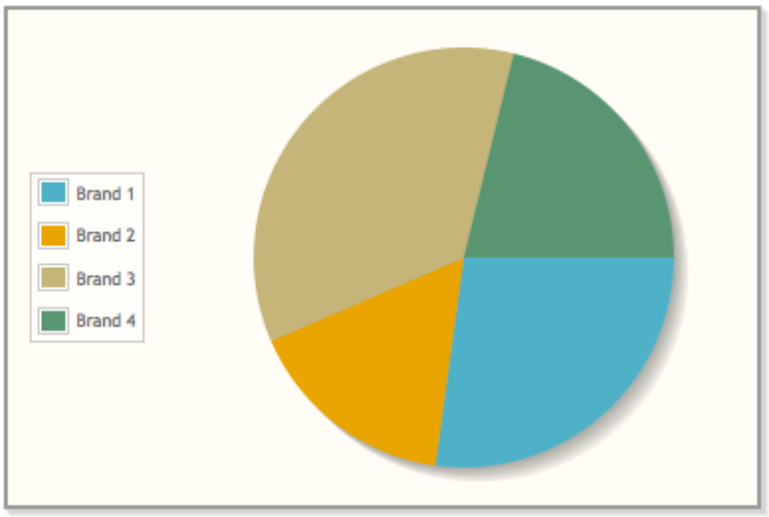
Name	Default	Type	Description
shadow	TRUE	Boolean	Shows shadow or not.
showDataLabels	FALSE	Boolean	Displays data on each slice.
dataFormat	percent	String	Format of data labels.
legendCols	1	Integer	Column count of legend.
legendRows	null	Integer	Row count of legend.

Getting started with PieChart

PieChart is created with an *org.primefaces.model.chart.PieChartModel* instance.

```
public class Bean {

	private PieChartModel model;

	public Bean() {
		model = new PieChartModel();
		model.set("Brand 1", 540);
		model.set("Brand 2", 325);
		model.set("Brand 3", 702);
		model.set("Brand 4", 421);
	}

	public PieChartModel getModel() {
		return model;
	}
}
```

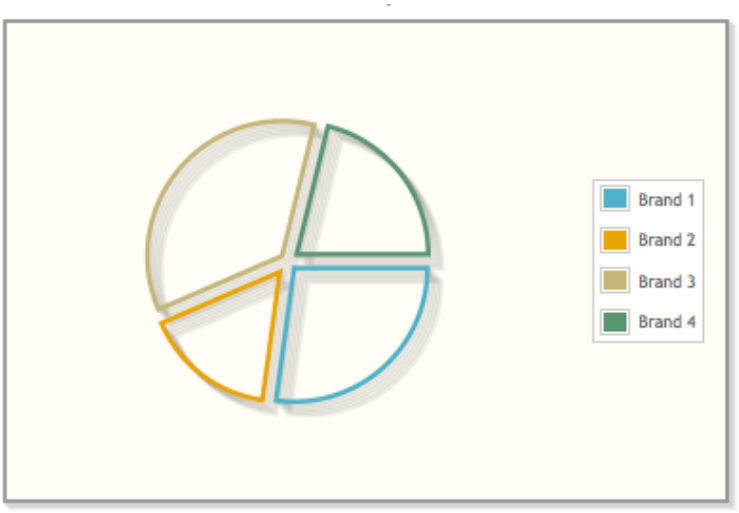
```
<p:pieChart value="#{bean.model}" legendPosition="w" />
```



Customization

PieChart can be customized using various options such as fill, sliceMargin and diameter, here is an example;

```
<p:pieChart value="#{bean.model}" legendPosition="e" sliceMargin="5"
     diameter="150" fill="false"/>
```



3.12.2 Line Chart

Line chart visualizes one or more series of data in a line graph.

Info

Tag	**lineChart**
Component Class	**org.primefaces.component.chart.line.LineChart**
Component Type	**org.primefaces.component.chart.LineChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.LineChartRenderer**
Renderer Class	**org.primefaces.component.chart.line.LineChartRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	ChartModel	Datasource to be displayed on the chart
widgetVar	null	String	Name of the client side widget
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
minY	null	Double	Minimum Y axis value.
maxY	null	Double	Maximum Y axis value.
minX	null	Double	Minimum X axis value.
maxX	null	Double	Maximum X axis value.
breakOnNull	FALSE	Boolean	Whether line segments should be broken at null value, fall will join point on either side of line.
seriesColors	null	String	Comma separated list of colors in hex format.

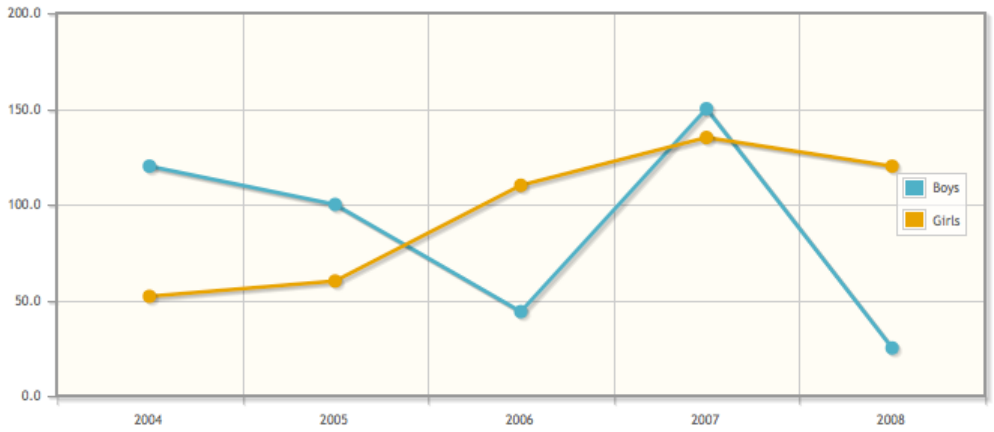
Name	Default	Type	Description
shadow	TRUE	Boolean	Shows shadow or not.
fill	FALSE	Boolean	Whether to fill under lines.
stacked	FALSE	Boolean	Whether to stack series.
showMarkers	TRUE	Boolean	Displays markers at data points.
fillToZero	FALSE	Boolean	True will force filled series to fill toward zero.
xaxisLabel	null	String	Label of the x-axis.
yaxisLabel	null	String	Label of the y-axis.
xaxisAngle	null	Integer	Angle of the x-axis ticks.
yaxisAngle	null	Integer	Angle of the y-axis ticks.
legendCols	1	Integer	Column count of legend.
legendRows	null	Integer	Row count of legend.

Getting started with LineChart

LineChart is created with an *org.primefaces.model.chart.CartesianChartModel* instance.

```
public class Bean {
      private CartesianChartModel model;

      public ChartBean() {
            model = new CartesianChartModel();

            ChartSeries boys = new ChartSeries();
            boys.setLabel("Boys");

            boys.set("2004", 120);
            boys.set("2005", 100);

            ChartSeries girls = new ChartSeries();
            girls.setLabel("Girls");

            girls.set("2004", 52);
            girls.set("2005", 60);

            model.addSeries(boys);
            model.addSeries(girs);
      }

      public CartesianChartModel getModel() { return model; }
}
```

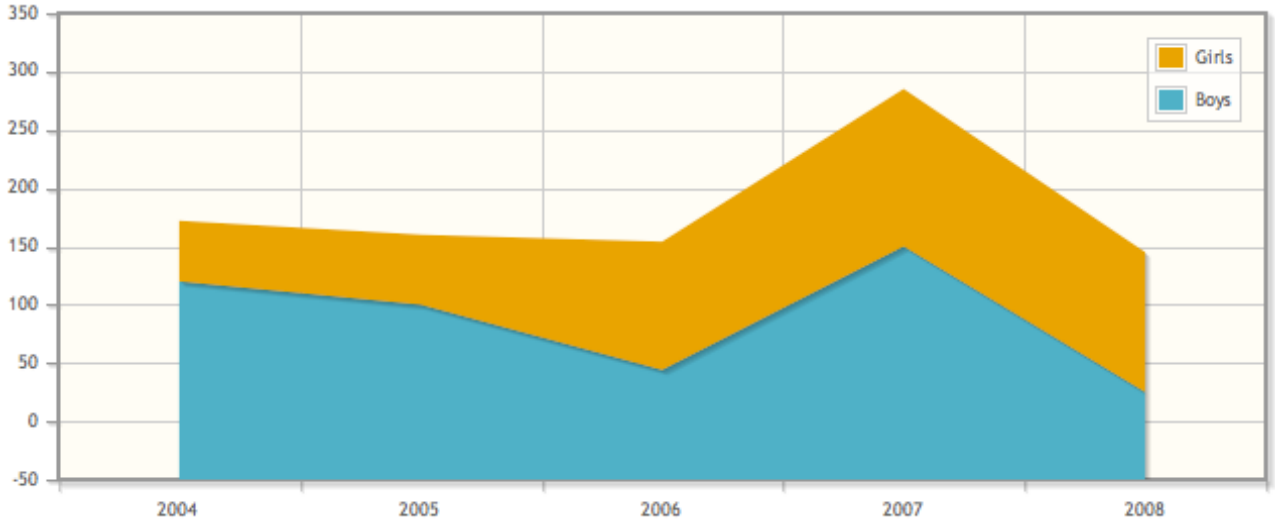
```
<p:lineChart value="#{chartBean.model}" legendPosition="e" />
```



AreaChart

AreaCharts is implemented by enabling stacked and fill options.

```
<p:lineChart value="#{bean.model}" legendPosition="ne"
                  fill="true" stacked="true"/>
```



3.12.3 Bar Chart

Bar chart visualizes one or more series of data using bars.

Info

Tag	**barChart**
Component Class	**org.primefaces.component.chart.bar.BarChart**
Component Type	**org.primefaces.component.chart.BarChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.BarChartRenderer**
Renderer Class	**org.primefaces.component.chart.bar.BarChartRenderer**

Attributes

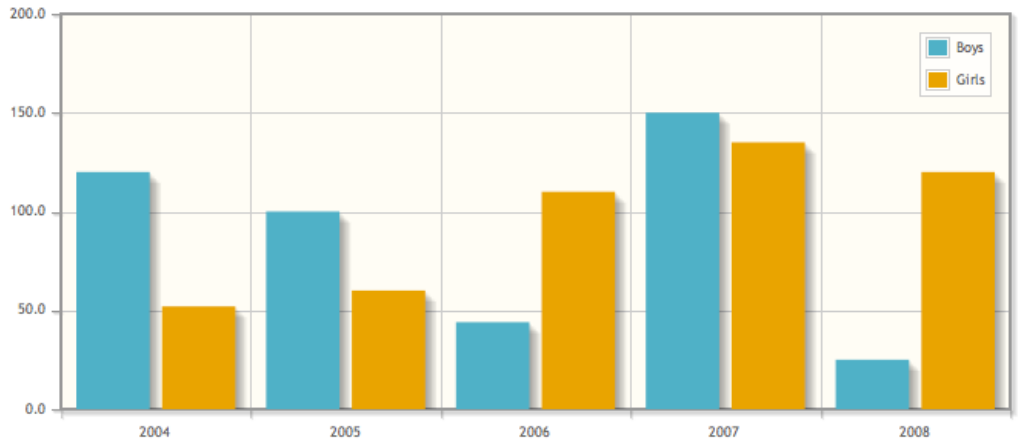
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
barPadding	8	Integer	Padding of bars.
barMargin	10	Integer	Margin of bars.
orientation	vertical	String	Orientation of bars, valid values are "vertical" and "horizontal".
stacked	FALSE	Boolean	Enables stacked display of bars.
min	null	Double	Minimum boundary value.
max	null	Double	Maximum boundary value.

Name	Default	Type	Description
breakOnNull	FALSE	Boolean	Whether line segments should be broken at null value, fall will join point on either side of line.
seriesColors	null	String	Comma separated list of colors in hex format.
shadow	TRUE	Boolean	Shows shadow or not.
enhancedLegend	TRUE	Boolean	Specifies interactive legend.
xaxisLabel	null	String	Label of the x-axis.
yaxisLabel	null	String	Label of the y-axis.
xaxisAngle	null	Integer	Angle of the x-axis ticks.
yaxisAngle	null	Integer	Angle of the y-axis ticks.
legendCols	1	Integer	Column count of legend.
legendRows	null	Integer	Row count of legend.

Getting Started with Bar Chart

BarChart is created with an *org.primefaces.model.chart.CartesianChartModel* instance. Reusing the same model sample from lineChart section;

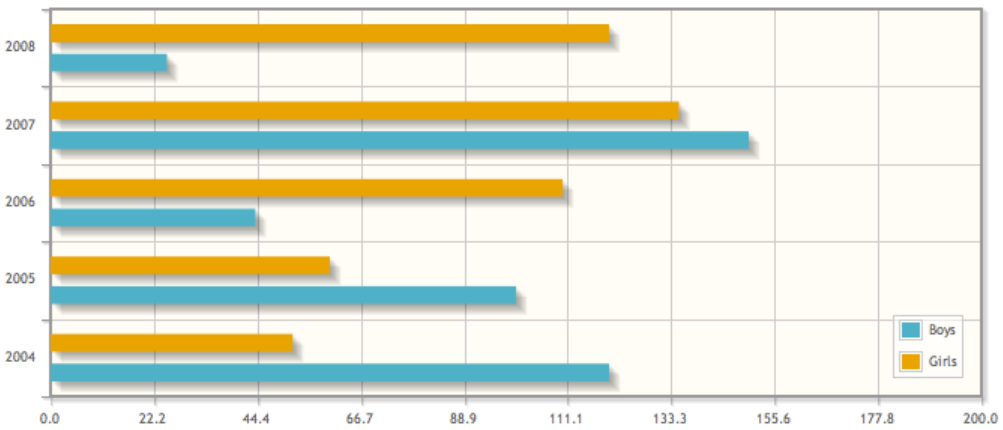
```
<p:barChart value="#{bean.model}" legendPosition="ne" />
```



Orientation

Bars can be displayed horizontally using the orientation attribute.

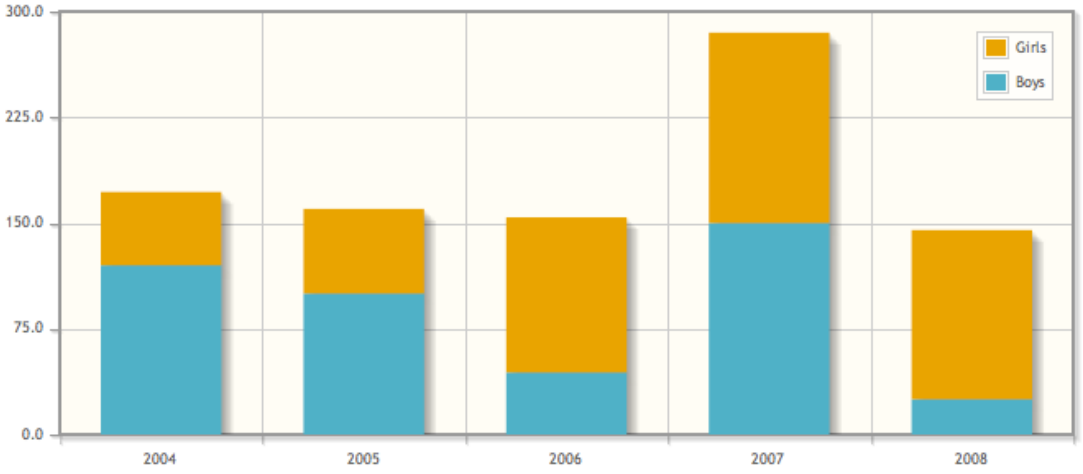
```
<p:barChart value="#{bean.model}" legendPosition="ne" orientation=”horizontal” />
```



Stacked BarChart

Enabling stacked option displays bars in stacked format..

```
<p:barChart value="#{bean.model}" legendPosition="se" stacked=”true” />
```



3.12.4 Donut Chart

DonutChart is a combination of pie charts.

Info

Tag	**donutChart**
Component Class	**org.primefaces.component.chart.donut.DonutChart**
Component Type	**org.primefaces.component.chart.DonutChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.DonutChartRenderer**
Renderer Class	**org.primefaces.component.chart.donut.DonutChartRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
seriesColors	null	String	Comma separated list of colors in hex format.
sliceMargin	0	Integer	Gap between slices.
fill	TRUE	Boolean	Render solid slices.
shadow	TRUE	Boolean	Shows shadow or not.
showDataLabels	FALSE	Boolean	Displays data on each slice.
dataFormat	percent	String	Format of data labels.
legendCols	1	Integer	Column count of legend.

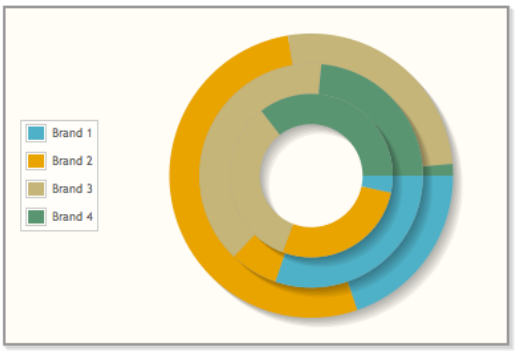
Name	Default	Type	Description
legendRows	null	Integer	Row count of legend.

Getting started with DonutChart

PieChart is created with an *org.primefaces.model.chart.DonutChartModel* instance.

```
public class Bean {

	private DonutChart model;

	public Bean() {
		model = new DonutChart();

		Map<String, Number> circle1 = new LinkedHashMap<String, Number>();
		circle1.put("Brand 1", 150);
		circle1.put("Brand 2", 400);
		circle1.put("Brand 3", 200);
		circle1.put("Brand 4", 10);
		donutModel.addCircle(circle1);

		Map<String, Number> circle2 = new LinkedHashMap<String, Number>();
		circle2.put("Brand 1", 540);
		circle2.put("Brand 2", 125);
		circle2.put("Brand 3", 702);
		circle2.put("Brand 4", 421);
		donutModel.addCircle(circle2);

		Map<String, Number> circle3 = new LinkedHashMap<String, Number>();
		circle3.put("Brand 1", 40);
		circle3.put("Brand 2", 325);
		circle3.put("Brand 3", 402);
		circle3.put("Brand 4", 421);
		donutModel.addCircle(circle3);
	}

	public DonutChart getModel() { return model; }
}
```

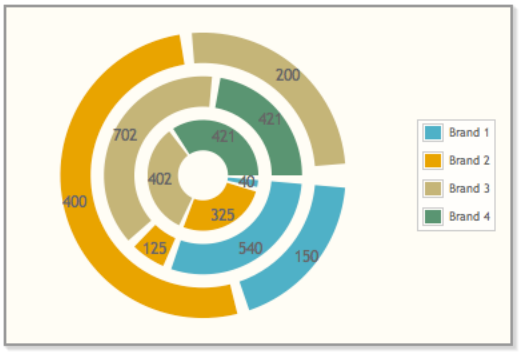
```
<p:donutChart value="#{bean.model}" legendPosition="w" />
```



Customization

DonutChart can be customized using various options;

```
<p:donutChart model="#{bean.model}" legendPosition="e" sliceMargin="5"
      showDataLabels="true" dataFormat="value" shadow="false"/>
```



3.12.5 Bubble Chart

BubbleChart visualizes entities that are defined in terms of three distinct numeric values.

Info

Tag	**bubbleChart**
Component Class	**org.primefaces.component.chart.bubble.BubbleChart**
Component Type	**org.primefaces.component.chart.BubbleChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.BubbleChartRenderer**
Renderer Class	**org.primefaces.component.chart.bubble.BubbleChartRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
shadow	TRUE	Boolean	Shows shadow or not.
seriesColors	null	String	Comma separated list of colors in hex format.
bubbleGradients	FALSE	Boolean	Enables gradient fills instead of flat colors.
bubbleAlpha	70	Integer	Alpha transparency of a bubble.
showLabels	TRUE	Boolean	Displays labels on buttons.
xaxisLabel	null	String	Label of the x-axis.
yaxisLabel	null	String	Label of the y-axis.

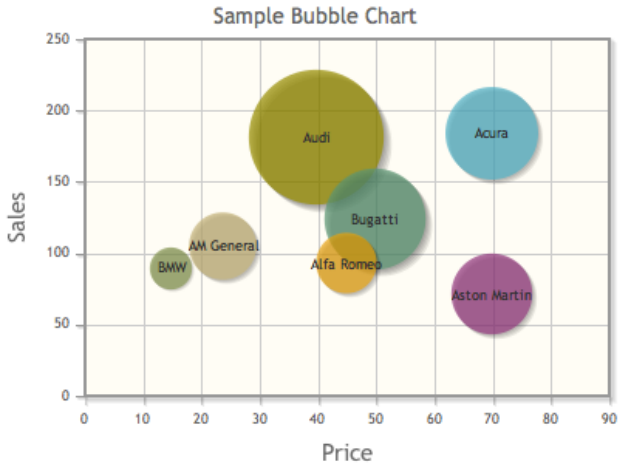
Name	Default	Type	Description
xaxisAngle	null	Integer	Angle of the x-axis ticks.
yaxisAngle	null	Integer	Angle of the y-axis ticks.
legendCols	1	Integer	Column count of legend.
legendRows	null	Integer	Row count of legend.

Getting started with BubbleChart

PieChart is created with an *org.primefaces.model.chart.BubbleChartModel* instance.

```
public class Bean {

      private BubbleChartModel model;

      public Bean() {
          bubbleModel = new BubbleChartModel();

           bubbleModel.addBubble(new BubbleChartSeries("Acura", 70, 183,55));
           bubbleModel.addBubble(new BubbleChartSeries("Alfa Romeo", 45, 92, 36));
           bubbleModel.addBubble(new BubbleChartSeries("AM General", 24, 104, 40));
           bubbleModel.addBubble(new BubbleChartSeries("Bugatti", 50, 123, 60));
           bubbleModel.addBubble(new BubbleChartSeries("BMW", 15, 89, 25));
           bubbleModel.addBubble(new BubbleChartSeries("Audi", 40, 180, 80));
           bubbleModel.addBubble(new BubbleChartSeries("AstonMartin", 70, 70, 48));
      }

      public BubbleChartModel getModel() {
             return model;
      }
}
```

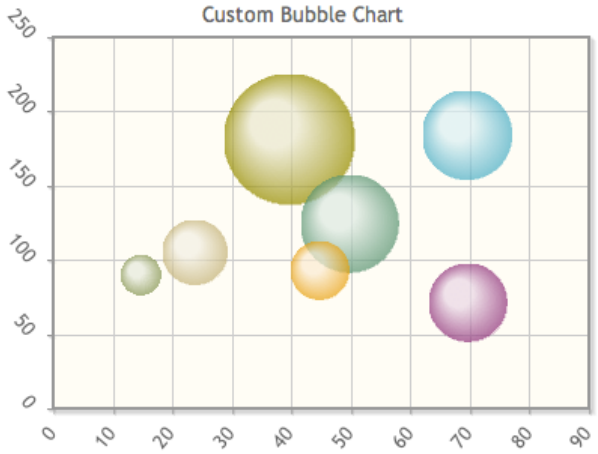
```
<p:bubbleChart value="#{bean.model}" xaxisLabel="Price" yaxisLabel="Sales"
                  title="Sample Bubble Chart"/>
```



Customization

BubbleChart can be customized using various options;

```
<p:bubbleChart value="#{bean.model}" bubbleGradients="true" shadow="false"
                    title="Custom Bubble Chart" showLabels="false" bubbleAlpha="100"
                    xaxisAngle="-50" yaxisAngle="50" />
```



3.12.6 Ohlc Chart

An open-high-low-close chart is a type of graph typically used to visualize movements in the price of a financial instrument over time.

Info

Tag	**ohlcChart**
Component Class	**org.primefaces.component.chart.ohlc.OhlcChart**
Component Type	**org.primefaces.component.chart.OhlcChart**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.OhlcChartRenderer**
Renderer Class	**org.primefaces.component.chart.ohlc.OhlcChartRenderer**

Attributes

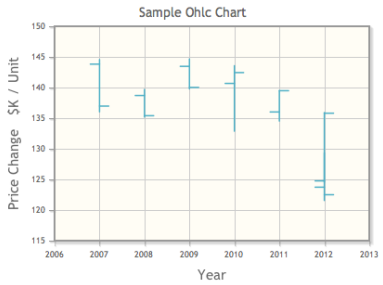
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
seriesColors	null	String	Comma separated list of colors in hex format.
candleStick	FALSE	Boolean	Enables candle stick display mode.
xaxisLabel	null	String	Label of the x-axis.
yaxisLabel	null	String	Label of the y-axis.
xaxisAngle	null	Integer	Angle of the x-axis ticks.
yaxisAngle	null	Integer	Angle of the y-axis ticks.

Getting started with OhlcChart

OhlcChart is created with an *org.primefaces.model.chart.OhlcChartModel* instance.

```
public class Bean {

    private OhlcChartModel model;

    public Bean() {
       model = new OhlcChartModel();

       ohlcModel.addRecord(new OhlcChartSeries(2007,143.82,144.56,136.04,136.97));
       ohlcModel.addRecord(new OhlcChartSeries(2008,138.7,139.68,135.18,135.4));
       ohlcModel.addRecord(new OhlcChartSeries(2009,143.46,144.66,139.79,140.02));
       ohlcModel.addRecord(new OhlcChartSeries(2010,140.67,143.56,132.88,142.44));
       ohlcModel.addRecord(new OhlcChartSeries(2011,136.01,139.5,134.53,139.48));
       ohlcModel.addRecord(new OhlcChartSeries(2012,124.76,135.9,124.55,135.81));
       ohlcModel.addRecord(new OhlcChartSeries(2012,123.73,129.31,121.57,122.5));
    }

    //getter
}
```

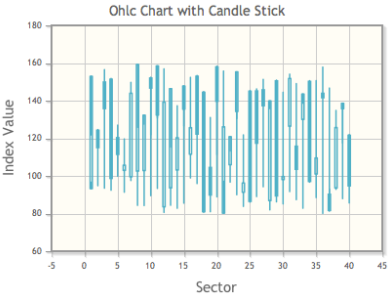
```
<p:ohlcChart value="#{bean.model}" xaxisLabel="Year"
             yaxisLabel="Price Change $K/Unit"  title="Sample Ohlc Chart"/>
```



CandleStick

OhlcChart can display data in candle stick format as well.

```
<p:ohlcChart value="#{bean.model}" xaxisLabel="Sector" yaxisLabel="Index Value"
             title="Ohlc Chart with Candle Stick" />
```



3.12.7 MeterGauge Chart

MeterGauge chart visualizes data on a meter gauge display.

Info

Tag	**meterGaugeChart**
Component Class	**org.primefaces.component.chart.metergauge.MeterGaugeChart**
Component Type	**org.primefaces.component.chart.MeterGauge**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.chart.MeterGaugeChartRenderer**
Renderer Class	**org.primefaces.component.chart.metergauge.MeterGaugeChartRenderer**

Attributes

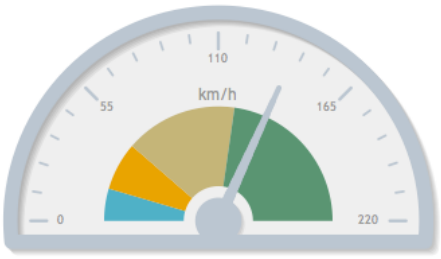
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	null	ChartModel	Datasource to be displayed on the chart
style	null	String	Inline style of the chart.
styleClass	null	String	Style class of the chart.
title	null	String	Title of the chart.
legendPosition	null	String	Position of the legend.
seriesColors	null	String	Comma separated list of colors in hex format.
enhancedLegend	TRUE	Boolean	Specifies interactive legend.
showTickLabels	TRUE	Boolean	Displays ticks around gauge.
labelHeightAdjust	-25	Integer	Number of pixels to offset the label up and down.
intervalOuterRadius	85	Integer	Radius of the outer circle of the internal ring.

Getting started with MeterGaugeChart

PieChart is created with an *org.primefaces.model.chart.MeterGaugeChartModel* instance.

```
public class Bean {

      private MeterGaugeChartModel model;

      public Bean() {
            List<Number> intervals = new ArrayList<Number>(){{
                  add(20);
                  add(50);
                  add(120);
                  add(220);
        }};

            model = new MeterGaugeChartModel("km/h", 140, intervals);
      }

      public MeterGaugeChartModel getModel() {
            return model;
      }
}
```

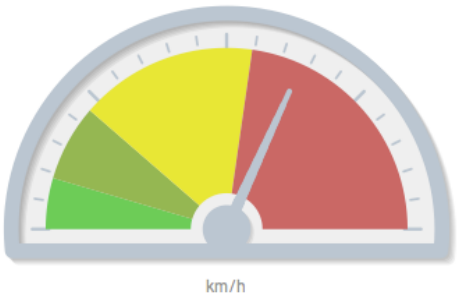
```
<p:meterGaugeChart value="#{bean.model}" />
```



Customization

MeterGaugeChart can be customized using various options;

```
<p:meterGaugeChart value="#{bean.model}" showTickLabels="false"
            labelHeightAdjust="110" intervalOuterRadius="110"
            seriesColors="66cc66, 93b75f, E7E658, cc6666" />
```



3.12.8 Skinning Charts

Charts are built on top of jqlot javascript library that uses a canvas tag and can be styled using regular css. Following is the list of style classes;

Style Class	Applies
.jqplot-target	Plot target container.
.jqplot-axis	Axes.
.jqplot-xaxis	Primary x-axis.
.jqplot-yaxis	Primary y-axis.
.jqplot-x2axis, .jqplot-x3axis ...	2nd, 3rd ... x-axis.
.jqplot-y2axis, .jqplot-y3axis ...	2nd, 3rd ... y-axis.
.jqplot-axis-tick	Axis ticks.
.jqplot-xaxis-tick	Primary x-axis ticks.
.jqplot-x2axis-tick	Secondary x-axis ticks.
.jqplot-yaxis-tick	Primary y-axis-ticks.
.jqplot-y2axis-tick	Seconday y-axis-ticks.
table.jqplot-table-legend	Legend table.
.jqplot-title	Title of the chart.
.jqplot-cursor-tooltip	Cursor tooltip.
.jqplot-highlighter-tooltip	Highlighter tooltip.
div.jqplot-table-legend-swatch	Colors swatch of the legend.

Additionally *style* and *styleClass* options of charts apply to the container element of charts, use these attribute to specify the dimensions of a chart.

```
<p:pieChart value="#{bean.model}" style=”width:320px;height:200px” />
```

In case you'd like to change the colors of series, use the *seriesColors* options.

```
<p:pieChart value="#{bean.model}" seriesColors="66cc66, 93b75f, E7E658, cc6666" />
```

3.12.9 Ajax Behavior Events

itemSelect is one and only ajax behavior event of charts, this event is triggered when a series of a chart is clicked. In case you have a listener defined, it'll be executed by passing an *org.primefaces.event.ItemSelectEvent* instance.

Example above demonstrates how to display a message about selected series in chart.

```
<p:pieChart value="#{bean.model}">
      <p:ajax event="itemSelect" listener="#{bean.itemSelect}" update="msg" />
</p:pieChart>

<p:growl id="msg" />
```

```
public class Bean implements Serializable {

      //Data creation omitted

      public void itemSelect(ItemSelectEvent event) {
            FacesMessage msg = new FacesMessage();
            msg.setSummary("Item Index: " + event.getItemIndex());
            msg.setDetail("Series Index:" + event.getSeriesIndex());

            FacesContext.getCurrentInstance().addMessage(null, msg);
      }
}
```

3.12.10 Charting Tips

jqPlot

Charts components use jqPlot as the underlying charting engine which uses a canvas element under the hood with support for IE.

jFreeChart

If you like to use static image charts instead of canvas based charts, see the JFreeChart integration example at graphicImage section. Note that static images charts are not rich as PrimeFaces chart components and you need to know about jFreeChart apis to create the charts.

3.13 Collector

Collector is a simple utility to manage collections declaratively.

Info

Tag	**collector**
ActionListener Class	**org.primefaces.component.collector.Collector**

Attributes

Name	Default	Type	Description
value	null	Object	Value to be used in collection operation
addTo	null	java.util.Collection	Reference to the Collection instance
removeFrom	null	java.util.Collection	Reference to the Collection instance

Getting started with Collector

Collector requires a collection and a value to work with. It's important to override equals and hashCode methods of the value object to make collector work.

```
public class BookBean {
      private Book book = new Book();
      private List<Book> books;

      public CreateBookBean() {
            books = new ArrayList<Book>();
      }

      public String createNew() {
            book = new Book();  //reset form
            return null;
      }

      //getters and setters
}
```

```
<p:commandButton value="Add" action="#{bookBean.createNew}">
      <p:collector value="#{bookBean.book}"  addTo="#{bookBean.books}" />
</p:commandButton>
```

```
<p:commandLink value="Remove">
      <p value="#{book}" removeFrom="#{createBookBean.books}" />
</p:commandLink>
```

3.14 Color Picker

ColorPicker is an input component with a color palette.

Info

Tag	**colorPicker**
Component Class	**org.primefaces.component.colorpicker.ColorPicker**
Component Type	**org.primefaces.component.ColorPicker**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ColorPickerRenderer**
Renderer Class	**org.primefaces.component.colorpicker.ColorPickerRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.

Name	Default	Type	Description
required	FALSE	Boolean	Marks component as required.
validator	null	MethodExpr	A method expression that refers to a method for validation the input.
valueChangeListener	null	ValueChange Listener	A method binding expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
mode	popup	String	Display mode, valid values are "popup" and "inline".
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.

Getting started with ColorPicker

ColorPicker's value should be a hex string.

```
public class Bean {

	private String color;

	public String getColor() {
		return this.color;
	}

	public void setColor(String color) {
		this.color = color;
	}
}
```

```
<p:colorPicker value="#{bean.color}" />
```

Display Mode

ColorPicker has two modes, default mode is *popup* and other available option is *inline*.

```
<p:colorPicker value="#{bean.color}" mode="inline"/>
```

Skinning

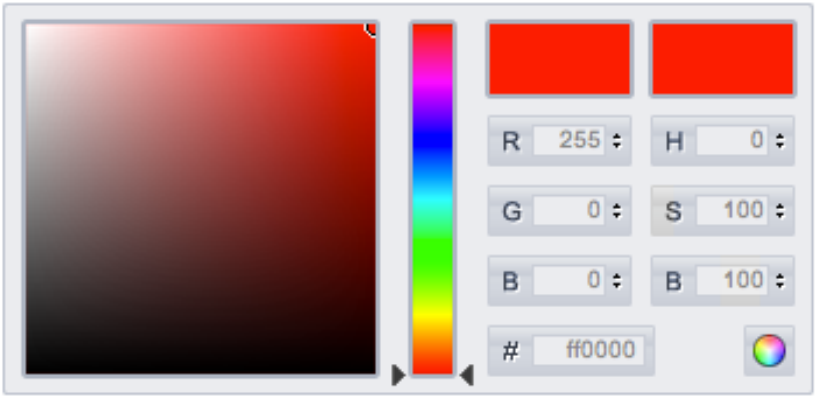
ColorPicker resides in a container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-colorpicker	Container element.
.ui-colorpicker_color	Background of gradient.
.ui-colorpicker_hue	Hue element.
.ui-colorpicker_new_color	New color display.
.ui-colorpicker_current_color	Current color display.
.ui-colorpicker-rgb-r	Red input.
.ui-colorpicker-rgb-g	Greed input.
.ui-colorpicker-rgb-b	Blue input.
.ui-colorpicker-rgb-h	Hue input.
.ui-colorpicker-rgb-s	Saturation input.
.ui-colorpicker-rgb-b	Brightness input.
.ui-colorpicker-rgb-hex	Hex input.

Example below changes the skin of color picker to a silver look and feel.

```
<p:colorPicker value="#{bean.color}" styleClass="silver" />
```



```
 .silver .ui-colorpicker-container {
      background-image: url(silver_background.png);
 }

.silver .ui-colorpicker_hex {
      background-image: url(silver_hex.png);
 }

.silver .ui-colorpicker_rgb_r {
      background-image: url(silver_rgb_r.png);
}

.silver .ui-colorpicker_rgb_g {
      background-image: url(silver_rgb_g.png);
}

.silver .ui-colorpicker_rgb_b {
      background-image: url(silver_rgb_b.png);
}

.silver .ui-colorpicker_hsb_s {
      background-image: url(silver_hsb_s.png);
}

.silver .ui-colorpicker_hsb_h {
      background-image: url(silver_hsb_h.png);
}

.silver .ui-colorpicker_hsb_b {
      background-image: url(silver_hsb_b.png);
}

.silver .ui-colorpicker_submit {
      background-image: url(silver_submit.png);
}
```

3.15 Column

Column is an extended version of the standard column used by various PrimeFaces components like datatable, treetable and more.

Info

Tag	**column**
Component Class	**org.primefaces.component.column.Column**
Component Type	**org.primefaces.component.Column**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Inline style of the column.
styleClass	null	String	Style class of the column.
sortBy	null	ValueExpr	Property to be used for sorting.
sortFunction	null	String/MethodExpr	Custom pluggable sortFunction.
filterBy	null	ValueExpr	Property to be used for filtering.
filterStyle	null	String	Inline style of the filter element
filterStyleClass	null	String	Style class of the filter element
filterOptions	null	Object	A collection of selectitems for filter dropdown.
filterMatchMode	startsWith	String	Match mode for filtering.
rowspan	1	Integer	Defines the number of rows the column spans.
colspan	1	Integer	Defines the number of columns the column spans.
headerText	null	String	Shortcut for header facet.
footerText	null	String	Shortcut for footer facet.
selectionMode	null	String	Enables selection mode.

Name	Default	Type	Description
disabledSelection	FALSE	Boolean	Disables row selection.
filterMaxLength	null	Integer	Maximum number of characters for an input filter.
resizable	TRUE	Boolean	Specifies resizable feature at column level. Datatable's resizableColumns must be enabled to use this option.
width	null	Integer	Width in pixels.

Note

As column is a reused component, not all attributes of column are implemented by the components that use column, for example filterBy is only used by datatable whereas sortBy is used by datatable and sheet.

Getting Started with Column

As column is a reused component, see documentation of components that use a column.

3.16 Columns

Columns is used by datatable to create columns programmatically.

Info

Tag	**columns**
Component Class	**org.primefaces.component.column.Columns**
Component Type	**org.primefaces.component.Columns**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data to represent columns.
var	null	String	Name of iterator to access a column.
columnIndexVar	null	String	Name of iterator to access a column index.
style	null	String	Inline style of the column.
styleClass	null	String	Style class of the column.

Getting Started with Columns

See dynamic columns section in datatable documentation for detailed information.

3.17 ColumnGroup

ColumnGroup is used by datatable for column grouping.

Info

Tag	**columnGroup**
Component Class	**org.primefaces.component.columngroup.ColumnGroup**
Component Type	**org.primefaces.component. ColumnGroup**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
type	null	String	Type of group, valid values are "header" and "footer".

Getting Started with ColumnGroup

See grouping section in datatable documentation for detailed information.

3.18 CommandButton

CommandButton is an extended version of standard commandButton with ajax and theming.

Info

Tag	**commandButton**
Component Class	**org.primefaces.component.commandbutton.CommandButton**
Component Type	**org.primefaces.component.CommandButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CommandButtonRenderer**
Renderer Class	**org.primefaces.component.commandbutton.CommandButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label for the button
action	null	MethodExpr/ String	A method expression or a String outcome that'd be processed when button is clicked.
actionListener	null	MethodExpr	An actionlistener that'd be processed when button is clicked.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
type	submit	String	Sets the behavior of the button.
ajax	TRUE	Boolean	Specifies the submit mode, when set to true(default), submit would be made with Ajax.
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component(s) to process partially instead of whole view.
update	null	String	Component(s) to be updated with ajax.

Name	Default	Type	Description
onstart	null	String	Client side callback to execute before ajax request is begins.
oncomplete	null	String	Client side callback to execute when ajax request is completed.
onsuccess	null	String	Client side callback to execute when ajax request succeeds.
onerror	null	String	Client side callback to execute when ajax request fails.
global	TRUE	Boolean	Defines whether to trigger ajaxStatus or not.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
style	null	String	Inline style of the button element.
styleClass	null	String	StyleClass of the button element.
onblur	null	String	Client side callback to execute when button loses focus.
onchange	null	String	Client side callback to execute when button loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when button is clicked.
ondblclick	null	String	Client side callback to execute when button is double clicked.
onfocus	null	String	Client side callback to execute when button receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over button.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over button.
onkeyup	null	String	Client side callback to execute when a key is released over button.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over button.
onmousemove	null	String	Client side callback to execute when a pointer button is moved within button.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from button.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto button.
onmouseup	null	String	Client side callback to execute when a pointer button is released over button.
onselect	null	String	Client side callback to execute when text within button is selected by user.

Name	Default	Type	Description
accesskey	null	String	Access key that when pressed transfers focus to the button.
alt	null	String	Alternate textual description of the button.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables the button.
image	null	String	Style class for the button icon. (deprecated: use icon)
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
tabindex	null	Integer	Position of the button element in the tabbing order.
title	null	String	Advisory tooltip information.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
icon	null	String	Icon of the button as a css class.
iconPos	left	String	Position of the icon.
inline	FALSE	String	Used by PrimeFaces mobile only.
escape	TRUE	Boolean	Defines whether label would be escaped or not.
widgetVar	null	String	Name of the client side widget.

Getting started with CommandButton

CommandButton usage is similar to standard commandButton, by default commandButton submits its enclosing form with ajax.

```
<p:commandButton value="Save" actionListener="#{bookBean.saveBook}" />
```

```
public class BookBean {

   public void saveBook() {
      //Save book
   }
}
```

Reset Buttons

Reset buttons do not submit the form, just resets the form contents.

```
<p:commandButton type="reset" value="Reset" />
```

Push Buttons

Push buttons are used to execute custom javascript without causing an ajax/non-ajax request. To create a push button set type as "button".

```
<p:commandButton type="button" value="Alert" onclick="alert('Prime')" />
```

AJAX and Non-AJAX

CommandButton has built-in ajax capabilities, ajax submit is enabled by default and configured using *ajax* attribute. When ajax attribute is set to false, form is submitted with a regular full page refresh.

The *update* attribute is used to partially update other component(s) after the ajax response is received. Update attribute takes a comma or white-space separated list of JSF component ids to be updated. Basically any JSF component, not just PrimeFaces components should be updated with the Ajax response.

In the following example, form is submitted with ajax and *display* outputText is updated with the ajax response.

```
<h:form>
      <h:inputText value="#{bean.text}" />
      <p:commandButton value="Submit" update="display"/>
      <h:outputText value="#{bean.text}" id="display" />
</h:form>
```

```
Tip: You can use the ajaxStatus component to notify users about the ajax request.
```

Icons

An icon on a button is provided using *icon* option. iconPos is used to define the position of the button which can be "left" or "right".

```
<p:commandButton value="With Icon" icon="disk"/>
<p:commandButton icon="disk"/>
```

.disk is a simple css class with a background property;

```
.disk {
      background-image: url('disk.png') !important;
}
```

You can also use the pre-defined icons from ThemeRoller like *ui-icon-search*.

Client Side API

Widget: *PrimeFaces.widget.CommandButton*

Method	Params	Return Type	Description
disable()	-	void	Disables button
enable()	-	void	Enables button

Skinning

CommandButton renders a button tag which *style* and *styleClass* applies.

Following is the list of structural style classes;

Style Class	Applies
.ui-button	Button element
.ui-button-text-only	Button element when icon is not used
.ui-button-text	Label of button

As skinning style classes are global, see the main Skinning section for more information.

3.19 CommandLink

CommandLink extends standard JSF commandLink with Ajax capabilities.

Info

Tag	**commandLink**
Component Class	**org.primefaces.component.commandlink.CommandLink**
Component Type	**org.primefaces.component.CommandLink**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.CommandLinkRenderer**
Renderer Class	**org.primefaces.component. commandlink.CommandLinkRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Href value of the rendered anchor.
action	null	MethodExpr/String	A method expression or a String outcome that'd be processed when link is clicked.
actionListener	null	MethodExpr	An actionlistener that'd be processed when link is clicked.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component(s) to process partially instead of whole view.
ajax	TRUE	Boolean	Specifies the submit mode, when set to true (default), submit would be made with Ajax.
update	null	String	Component(s) to be updated with ajax.
onstart	null	String	Client side callback to execute before ajax request is begins.

Name	Default	Type	Description
oncomplete	null	String	Client side callback to execute when ajax request is completed.
onsuccess	null	String	Client side callback to execute when ajax request succeeds.
onerror	null	String	Client side callback to execute when ajax request fails.
global	TRUE	Boolean	Defines whether to trigger ajaxStatus or not.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
style	null	String	Style to be applied on the anchor element
styleClass	null	String	StyleClass to be applied on the anchor element
onblur	null	String	Client side callback to execute when link loses focus.
onclick	null	String	Client side callback to execute when link is clicked.
ondblclick	null	String	Client side callback to execute when link is double clicked.
onfocus	null	String	Client side callback to execute when link receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over link.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over link.
onkeyup	null	String	Client side callback to execute when a key is released over link.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over link.
onmousemove	null	String	Client side callback to execute when a pointer button is moved within link.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from link.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto link.
onmouseup	null	String	Client side callback to execute when a pointer button is released over link.
accesskey	null	String	Access key that when pressed transfers focus to the link.
charset	null	String	Character encoding of the resource designated by this hyperlink.

Name	Default	Type	Description
coords	null	String	Position and shape of the hot spot on the screen for client use in image maps.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	null	Boolean	Disables the link
hreflang	null	String	Languae code of the resource designated by the link.
rel	null	String	Relationship from the current document to the anchor specified by the link, values are provided by a space-separated list of link types.
rev	null	String	A reverse link from the anchor specified by this link to the current document, values are provided by a space-separated list of link types.
shape	null	String	Shape of hot spot on the screen, valid values are default, rect, circle and poly.
tabindex	null	Integer	Position of the button element in the tabbing order.
target	null	String	Name of a frame where the resource targeted by this link will be displayed.
title	null	String	Advisory tooltip information.
type	null	String	Type of resource referenced by the link.

Getting Started with CommandLink

CommandLink is used just like the standard h:commandLink, difference is form is submitted with ajax by default.

```
public class BookBean {

   public void saveBook() {
      //Save book
   }
}
```

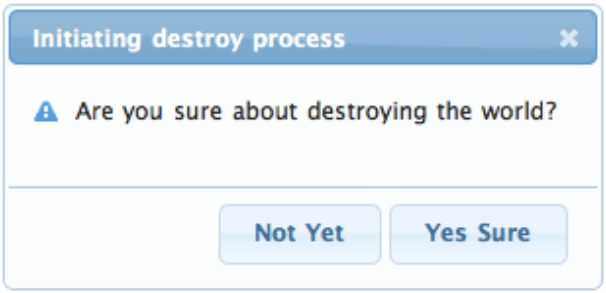
```
<p:commandLink actionListener="#{bookBean.saveBook}">
      <h:outputText value="Save" />
</p:commandLink>
```

Skinning

CommandLink renders an html anchor element that *style* and *styleClass* attributes apply.

3.20 ConfirmDialog

ConfirmDialog is a replacement to the legacy javascript confirmation box. Skinning, customization and avoiding popup blockers are notable advantages over classic javascript confirmation.



Info

Tag	**confirmDialog**
Component Class	**org.primefaces.component.confirmdialog.ConfirmDialog**
Component Type	**org.primefaces.component.ConfirmDialog**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ConfirmDialogRenderer**
Renderer Class	**org.primefaces.component.confirmdialog.ConfirmDialogRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
message	null	String	Text to be displayed in body.
header	null	String	Text for the header.
severity	null	String	Message severity for the displayed icon.
width	auto	Integer	Width of the dialog in pixels
height	auto	Integer	Width of the dialog in pixels
style	null	String	Inline style of the dialog container.

Name	Default	Type	Description
styleClass	null	String	Style class of the dialog container
closable	TRUE	Boolean	Defines if close icon should be displayed or not
appendToBody	FALSE	Boolean	Appends dialog as a child of document body.
visible	FALSE	Boolean	Whether to display confirm dialog on load.

Getting started with ConfirmDialog

ConfirmDialog has a simple client side api, *show()* and *hide()* functions are used to display and close the dialog respectively. You can call these functions to display a confirmation from any component like commandButton, commandLink, menuitem and more.

```
<h:form>
 <p:commandButton type="button" onclick="cd.show()" />

 <p:confirmDialog message="Are you sure about destroying the world?"
           header="Initiating destroy process" severity="alert"
           widgetVar="cd">

    <p:commandButton value="Yes Sure" actionListener="#{buttonBean.destroyWorld}"
                update="messages" oncomplete="confirmation.hide()"/>

    <p:commandButton value="Not Yet" onclick="confirmation.hide();" type="button" />

 </p:confirmDialog>
</h:form>
```

Message

Message can be defined in two ways, either via message option or message facet. Message facet is useful if you need to place custom content instead of simple text. Note that header can also be defined using the *header* attribute or the *header* facet.

```
<p:confirmDialog widgetVar="cd" header="Confirm">
     <f:facet name="message">
           <h:outputText value="Are you sure?" />
     </f:facet>

     //...
 </p:confirmDialog>
```

Severity

Severity defines the icon to display next to the message, default severity is *alert* and the other option is *info*.

Client Side API

Widget: *PrimeFaces.widget.ConfirmDialog*

Method	Params	Return Type	Description
show()	-	void	Displays dialog.
hide()	-	void	Closes dialog.

Skinning

ConfirmDialog resides in a main container element which *style* and *styleClass* options apply.

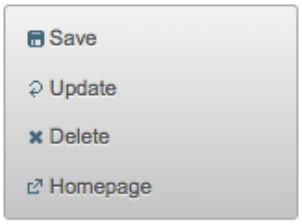
Following is the list of structural style classes;

Style Class	Applies
.ui-dialog	Container element of dialog
.ui-dialog-titlebar	Title bar
.ui-dialog-title	Header text
.ui-dialog-titlebar-close	Close icon
.ui-dialog-content	Dialog body
.ui-dialog-buttonpane	Footer button panel

As skinning style classes are global, see the main Skinning section for more information.

3.21 ContextMenu

ContextMenu provides an overlay menu displayed on mouse right-click event.



Info

Tag	**contextMenu**
Component Class	**org.primefaces.component.contextmenu.ContextMenu**
Component Type	**org.primefaces.component.ContextMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ContextMenuRenderer**
Renderer Class	**org.primefaces.component.contextmenu.ContextMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
for	null	String	Id of the component to attach to
style	null	String	Style of the main container element
styleClass	null	String	Style class of the main container element
model	null	MenuModel	Menu model instance to create menu programmatically.
nodeType	null	String	Specific type of tree nodes to attach to.
event	null	String	Event to bind contextMenu display, default is contextmenu aka right click.

Getting started with ContextMenu

ContextMenu is created with submenus and menuitems. Optional for attribute defines which component the contextMenu is attached to. When for is not defined, contextMenu is attached to the page meaning, right-click on anywhere on page will display the menu.

```
<p:contextMenu>
    <p:menuitem value="Save" actionListener="#{bean.save}" update="msg"/>
    <p:menuitem value="Delete" actionListener="#{bean.delete}" ajax="false"/>
    <p:menuitem value="Go Home" url="www.primefaces.org" target="_blank"/>
</p:contextMenu
```

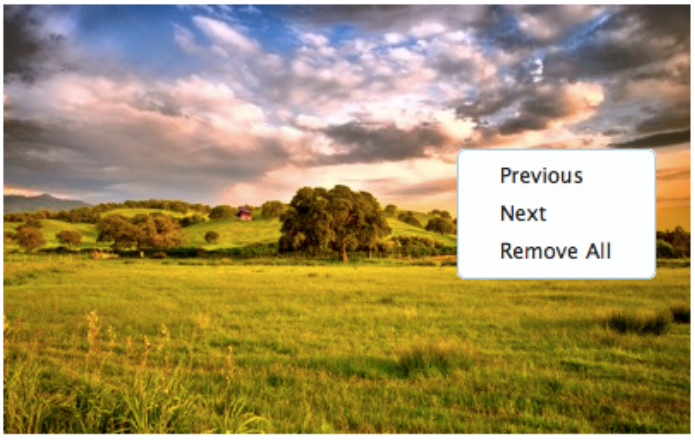
ContextMenu example above is attached to the whole page and consists of three different menuitems with different use cases. First menuitem triggers an ajax action, second one triggers a non-ajax action and third one is used for navigation.

Attachment

ContextMenu can be attached to any JSF component, this means right clicking on the attached component will display the contextMenu. Following example demonstrates an integration between contextMenu and imageSwitcher, contextMenu here is used to navigate between images.

```
<p:imageSwitch id="images" widgetVar="gallery" slideshowAuto="false">
      <p:graphicImage value="/images/nature1.jpg" />
      <p:graphicImage value="/images/nature2.jpg" />
      <p:graphicImage value="/images/nature3.jpg" />
      <p:graphicImage value="/images/nature4.jpg" />
</p:imageSwitch>

<p:contextMenu for="images">
      <p:menuitem value="Previous" url="#" onclick="gallery.previous()" />
      <p:menuitem value="Next" url="#" onclick="gallery.next()" />
</p:contextMenu>
```

Now right-clicking anywhere on an image will display the contextMenu like;



Data Components

Data components like datatable, tree and treeTable has special integration with context menu, see the documentation of these component for more information.

Dynamic Menus

ContextMenus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

ContextMenu resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-contextmenu	Container element of menu
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.22 Dashboard

Dashboard provides a portal like layout with drag&drop based reorder capabilities.

Info

Tag	**dashboard**
Component Class	**org.primefaces.component.dashboard.Dashboard**
Component Type	**org.primefaces.component.Dashboard**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DashboardRenderer**
Renderer Class	**org.primefaces.component.dashboard.DashboardRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
model	null	Dashboard Model	Dashboard model instance representing the layout of the UI.
disabled	FALSE	Boolean	Disables reordering feature.
style	null	String	Inline style of the dashboard container
styleClass	null	String	Style class of the dashboard container

Getting started with Dashboard

Dashboard is backed by a DashboardModel and consists of panel components.

```
<p:dashboard model="#{bean.model}">
      <p:panel id="sports">
            //Sports Content
      </p:panel>
      <p:panel id="finance">
            //Finance Content
      </p:panel>

      //more panels like lifestyle, weather, politics...
</p:dashboard>
```

Dashboard model simply defines the number of columns and the widgets to be placed in each column. See the end of this section for the detailed Dashboard API.

```
public class Bean {

      private DashboardModel model;

      public Bean() {
            model = new DefaultDashboardModel();
            DashboardColumn column1 = new DefaultDashboardColumn();
            DashboardColumn column2 = new DefaultDashboardColumn();
            DashboardColumn column3 = new DefaultDashboardColumn();

            column1.addWidget("sports");
            column1.addWidget("finance");
            column2.addWidget("lifestyle");
            column2.addWidget("weather");
            column3.addWidget("politics");

            model.addColumn(column1);
            model.addColumn(column2);
            model.addColumn(column3);
      }
}
```

State

Dashboard is a stateful component, whenever a widget is reordered dashboard model will be updated, by persisting the user changes so you can easily create a stateful dashboard.

Ajax Behavior Events

"reorder" is the one and only ajax behavior event provided by dashboard, this event is fired when dashboard panels are reordered. A defined listener will be invoked by passing an *org.primefaces.event.DashboardReorderEvent* instance containing information about reorder.

Following dashboard displays a message about the reorder event

```
<p:dashboard model="#{bean.model}">
      <p:ajax event="reorder" update="messages" listener="#{bean.handleReorder}" />
      //panels
 </p:dashboard>

<p:growl id="messages" />
```

```
public class Bean {

      ...

      public void handleReorder(DashboardReorderEvent event) {
            String widgetId = event.getWidgetId();
            int widgetIndex = event.getItemIndex();
            int columnIndex = event.getColumnIndex();
            int senderColumnIndex = event.getSenderColumnIndex();

            //Add facesmessage
      }
}
```

If a widget is reordered in the same column, *senderColumnIndex* will be null. This field is populated only when a widget is transferred to a column from another column. Also when the listener is invoked, dashboard has already updated it's model.

Disabling Dashboard

If you'd like to disable reordering feature, set *disabled* option to true.

```
<p:dashboard disabled="true" ...>
      //panels
 </p:dashboard>
```

Toggle, Close and Options Menu

Widgets presented in dashboard can be closable, toggleable and have options menu as well, dashboard doesn't implement these by itself as these features are already provided by the panel component. See panel component section for more information.

```
<p:dashboard model="#{dashboardBean.model}">
      <p:panel id="sports" closable="true" toggleable="true">
            //Sports Content
      </p:panel>
 </p:dashboard>
```

New Widgets

Draggable component is used to add new widgets to the dashboard. This way you can add new panels from outside of the dashboard.

```
<p:dashboard model="#{dashboardBean.model}" id="board">
      //panels
 </p:dashboard>

<p:panel id="newwidget" />

<p:draggable for="newwidget" helper="clone" dashboard="board" />
```

Skinning

Dashboard resides in a container element which style and styleClass options apply. Following is the list of structural style classes;

Style Class	Applies
.ui-dashboard	Container element of dashboard
.ui-dashboard-column	Each column in dashboard
div.ui-state-hover	Placeholder

As skinning style classes are global, see the main Skinning section for more information. Here is an example based on a different theme;

Tips

- Provide a column width using *ui-dashboard-column* style class otherwise empty columns might not receive new widgets.

Dashboard Model API

org.primefaces.model.DashboardModel (*org.primefaces.model.map.DefaultDashboardModel* is the default implementation)

Method	Description
void addColumn(DashboardColumn column)	Adds a column to the dashboard
List<DashboardColumn> getColumns()	Returns all columns in dashboard
int getColumnCount()	Returns the number of columns in dashboard
DashboardColumn getColumn(int index)	Returns the dashboard column at given index
void transferWidget(DashboardColumn from, DashboardColumn to, String widgetId, int index)	Relocates the widget identifed with widget id to the given index of the new column from old column.

org.primefaces.model.DashboardColumn (*org.primefaces.model.map.DefaultDashboardModel* is the default implementation)

Method	Description
void removeWidget(String widgetId)	Removes the widget with the given id
List<String> getWidgets()	Returns the ids of widgets in column
int getWidgetCount()	Returns the count of widgets in column
String getWidget(int index)	Returns the widget id with the given index
void addWidget(String widgetId)	Adds a new widget with the given id
void addWidget(int index, String widgetId)	Adds a new widget at given index
void reorderWidget(int index, String widgetId)	Updates the index of widget in column

3.23 DataExporter

DataExporter is handy for exporting data listed using a Primefaces Datatable to various formats such as excel, pdf, csv and xml.

Info

Tag	**dataExporter**
Tag Class	**org.primefaces.component.export.DataExporterTag**
ActionListener Class	**org.primefaces.component.export.DataExporter**

Attributes

Name	Default	Type	Description
type	null	String	Export type: "xls","pdf","csv", "xml"
target	null	String	Id of the datatable whose data to export.
fileName	null	String	Filename of the generated export file, defaults to datatable id.
pageOnly	FALSE	String	Exports only current page instead of whole dataset
excludeColumns	null	String	Comma separated list(if more than one) of column indexes to be excluded from export
preProcessor	null	MethodExpr	PreProcessor for the exported document.
postProcessor	null	MethodExpr	PostProcessor for the exported document.
encoding	UTF-8	Boolean	Character encoding to use
selectionOnly	FALSE	Boolean	When enabled, only selection would be exported.

Getting Started with DataExporter

DataExporter is nested in a UICommand component such as commandButton or commandLink. For pdf exporting **itext** and for xls exporting **poi** libraries are required in the classpath. Target must point to a PrimeFaces Datatable. Assume the table to be exported is defined as;

```
<p:dataTable id="tableId" ...>
     //columns
</p:dataTable>
```

Excel export

```
<p:commandButton value="Export as Excel" ajax="false">
     <p:dataExporter type="xls" target="tableId" fileName="cars"/>
</p:commandButton>
```

PDF export

```
<p:commandButton value="Export as PDF" ajax="false" >
     <p:dataExporter type="pdf" target="tableId" fileName="cars"/>
</p:commandButton>
```

CSV export

```
<p:commandButton value="Export as CSV" ajax="false" >
     <p:dataExporter type="csv" target="tableId" fileName="cars"/>
</p:commandButton>
```

XML export

```
<p:commandButton value="Export as XML" ajax="false" >
     <p:dataExporter type="xml" target="tableId" fileName="cars"/>
</p:commandLink>
```

PageOnly

By default dataExporter works on whole dataset, if you'd like export only the data displayed on current page, set pageOnly to true.

```
<p:dataExporter type="pdf" target="tableId" fileName="cars" pageOnly="true"/>
```

Excluding Columns

In case you need one or more columns to be ignored use *exludeColumns* option. Exporter below ignores first column, to exclude more than one column define the indexes as a comma separated string (excludeColumns="0,2,6").

```
<p:dataExporter type="pdf" target="tableId" excludeColumns="0"/>
```

Monitor Status

DataExport is a non-ajax process so ajaxStatus component cannot apply. See FileDownload Monitor Status section to find out how monitor export process. Same solution applies to data export as well.

Pre and Post Processors

Processors are handy to customize the exported document (e.g. add logo, caption ...). PreProcessors are executed before the data is exported and PostProcessors are processed after data is included in the document. Processors are simple java methods taking the document as a parameter.

Change Excel Table Header

First example of processors changes the background color of the exported excel's headers.

```
<h:commandButton value="Export as XLS">
      <p:dataExporter type="xls" target="tableId" fileName="cars"
                           postProcessor="#{bean.postProcessXLS}"/>
</h:commandButton>
```

```
public void postProcessXLS(Object document) {
      HSSFWorkbook wb = (HSSFWorkbook) document;
      HSSFSheet sheet = wb.getSheetAt(0);
      HSSFRow header = sheet.getRow(0);
      HSSFCellStyle cellStyle = wb.createCellStyle();
      cellStyle.setFillForegroundColor(HSSFColor.GREEN.index);
      cellStyle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

      for(int i=0; i < header.getPhysicalNumberOfCells();i++) {
            header.getCell(i).setCellStyle(cellStyle);
      }
}
```

Add Logo to PDF

This example adds a logo to the PDF before exporting begins.

```
<h:commandButton value="Export as PDF">
      <p:dataExporter type="xls" target="tableId" fileName="cars"
      preProcessor="#{bean.preProcessPDF}"/>
</h:commandButton>
```

```
public void preProcessPDF(Object document) throws IOException,
                           BadElementException, DocumentException {
      Document pdf = (Document) document;
      ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
      String logo = servletContext.getRealPath("") + File.separator + "images" +
File.separator + "prime_logo.png";
      pdf.add(Image.getInstance(logo));
}
```

3.24 DataGrid

DataGrid displays a collection of data in a grid layout.

Info

Tag	**dataGrid**
Component Class	**org.primefaces.component.datagrid.DataGrid**
Component Type	**org.primefaces.component.DataGrid**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DataGridRenderer**
Renderer Class	**org.primefaces.component.datagrid.DataGridRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.

Name	Default	Type	Description
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed
widgetVar	null	String	Name of the client side widget.
columns	3	Integer	Number of columns in grid.
paginator	FALSE	boolean	Enables pagination.
paginatorTemplate	null	String	Template of the paginator.
rowsPerPageTemplate	null	String	Template of the rowsPerPage dropdown.
currentPageReportTemplate	null	String	Template of the currentPageReport UI.
pageLinks	10	Integer	Maximum number of page links to display.
paginatorPosition	both	String	Position of the paginator.
paginatorAlwaysVisible	TRUE	Boolean	Defines if paginator should be hidden if total data count is less than number of rows per page.
style	null	String	Inline style of the datagrid.
styleClass	null	String	Style class of the datagrid.
rowIndexVar	null	String	Name of the iterator to refer each row index.

Getting started with the DataGrid

A list of cars will be used throughout the datagrid, datalist and datatable examples.

```
public class Car {

      private String model;
      private int year;
      private String manufacturer;
      private String color;
      ...

}
```

The code for CarBean that would be used to bind the datagrid to the car list.

```
public class CarBean {

      private List<Car> cars;

      public CarBean() {
            cars = new ArrayList<Car>();
            cars.add(new Car("myModel",2005,"ManufacturerX","blue"));
            //add more cars
      }

      public List<Car> getCars() {
            return cars;
      }
}
```

```
<p:dataGrid var="car" value="#{carBean.cars}" columns="3" rows="12">

      <p:column>
         <p:panel header="#{car.model}">
            <h:panelGrid columns="1">
                <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg"/>

                <h:outputText value="#{car.year}" />
            </h:panelGrid>
         </p:panel>
      </p:column>

</p:dataGrid>
```

This datagrid has 3 columns and 12 rows. As datagrid extends from standard UIData, rows correspond to the number of data to display not the number of rows to render so the actual number of rows to render is rows/columns = 4. As a result datagrid is displayed as;

Ajax Pagination

DataGrid has a built-in paginator that is enabled by setting paginator option to true.

```
<p:dataGrid var="car" value="#{carBean.cars}" columns="3" rows="12"
      paginator="true">
      ...
</p:dataGrid>
```

Paginator Template

Paginator is customized using paginatorTemplateOption that accepts various keys of UI controls. Note that this section applies to dataGrid, dataList and dataTable.

- FirstPageLink
- LastPageLink
- PreviousPageLink
- NextPageLink
- PageLinks
- CurrentPageReport
- RowsPerPageDropdown

Note that {RowsPerPageDropdown} has it's own template, options to display is provided via rowsPerPageTemplate attribute (e.g. rowsPerPageTemplate="9,12,15").

Also {CurrentPageReport} has it's own template defined with currentPageReportTemplate option. You can use {currentPage},{totalPages},{totalRecords},{startRecord},{endRecord} keyword within currentPageReportTemplate. Default is {currentPage} of{totalPages}.

Default UI is;

which corresponds to the following template.

"{FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink}"

Here are more examples based on different templates;

" {CurrentPageReport} {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropdown}"

" {PreviousPageLink} {CurrentPageReport} {NextPageLink}"

Paginator Position

Paginator can be positoned using *paginatorPosition* attribute in three different locations, "top", "bottom" or "both" (default).

Selecting Data

Selection of data displayed in datagrid is very similar to row selection in datatable, you can access the current data using the var reference. Important point is to place datagrid contents in a p:column which is a child of datagrid. Here is an example to demonstrate how to select data from datagrid and display within a dialog with ajax.

```
<h:form id="carForm">

 <p:dataGrid var="car" value="#{carBean.cars}" columns="3" rows="12">
      <p:column>
         <p:panel header="#{car.model}">
             <p:commandLink update="carForm:display" oncomplete="dlg.show()">
                    <f:setPropertyActionListener value="#{car}"
                                 target="#{carBean.selectedCar}"
                    <h:outputText value="#{car.model}" />
             </p:commandLink>
         </p:panel>
      </p:column>
 </p:dataGrid>

 <p:dialog modal="true" widgetVar="dlg">
      <h:panelGrid id="display" columns="2">
             <f:facet name="header">
                    <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg"/>
             </f:facet>
             <h:outputText value="Model:" />
             <h:outputText value="#{carBean.selectedCar.year}" />

             //more selectedCar properties
      </h:panelGrid>
 </p:dialog>

</h:form>
```

```
public class CarBean {

      private List<Car> cars;

      private Car selectedCar;

      //getters and setters
}
```

Client Side API

Widget: *PrimeFaces.widget.DataGrid*

Method	Params	Return Type	Description
getPaginator()	-	Paginator	Returns the paginator widget.

Skinning

DataGrid resides in a main div container which style and styleClass attributes apply.

Following is the list of structural style classes;

Class	Applies
.ui-datagrid	Main container element
.ui-datagrid-content	Content container.
.ui-datagrid-data	Table element containing data
.ui-datagrid-row	A row in grid
.ui-datagrid-column	A column in grid

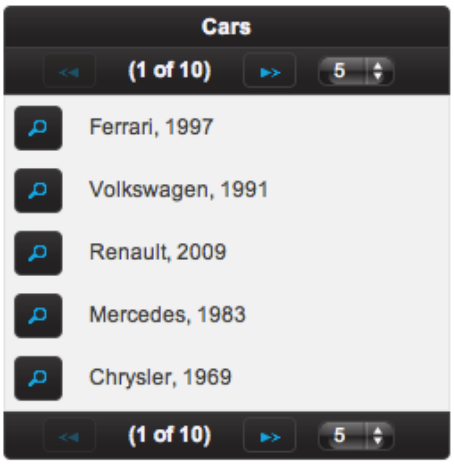
As skinning style classes are global, see the main Skinning section for more information.

Tips

- DataGrid supports lazy loading data via LazyDataModel, see DataTable lazy loading section.
- DataGrid provides two facets named *header* and *footer* that you can use to provide custom content at these locations.

3.25 DataList

DataList presents a collection of data in list layout with several display types.



Info

Tag	**dataList**
Component Class	**org.primefaces.component.datalist.DataList**
Component Type	**org.primefaces.component.DataList.DataListTag**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DataListRenderer**
Renderer Class	**org.primefaces.component.datalist.DataListRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed

Name	Default	Type	Description
type	unordered	String	Type of the list, valid values are "unordered", "ordered", "definition" and "none".
itemType	null	String	Specifies the list item type.
widgetVar	null	String	Name of the client side widget.
paginator	FALSE	boolean	Enables pagination.
paginatorTemplate	null	String	Template of the paginator.
rowsPerPageTemplate	null	String	Template of the rowsPerPage dropdown.
currentPageReportTemplate	null	String	Template of the currentPageReport UI.
pageLinks	10	Integer	Maximum number of page links to display.
paginatorPosition	both	String	Position of the paginator.
paginatorAlwaysVisible	TRUE	Boolean	Defines if paginator should be hidden if total data count is less than number of rows per page.
style	null	String	Inline style of the main container.
styleClass	Null	String	Style class of the main container.
rowIndexVar	null	String	Name of the iterator to refer each row index.
varStatus	null	String	Name of the exported request scoped variable to represent state of the iteration same as in ui:repeat varStatus.

Getting started with the DataList

Since DataList is a data iteration component, it renders it's children for each data represented with *var* option. See itemType section for more information about the possible values.

```
<p:dataList value="#{carBean.cars}" var="car" itemType="disc">
     #{car.manufacturer}, #{car.year}
</p:dataList>
```

Ordered Lists

DataList displays the data in unordered format by default, if you'd like to use ordered display set *type* option to "ordered".

```
<p:dataList value="#{carBean.cars}" var="car" type="ordered">
     #{car.manufacturer}, #{car.year}
</p:dataList>
```

Item Type

itemType defines the bullet type of each item. For ordered lists, following item types are available;

A	a	i
A. Ferrari, 1960 B. Renault, 1985 C. Ford, 2003 D. Audi, 1976 E. Opel, 1983 F. Ferrari, 1974 G. Chrysler, 1980 H. Audi, 1980 I. Chrysler, 1983	a. Ferrari, 1960 b. Renault, 1985 c. Ford, 2003 d. Audi, 1976 e. Opel, 1983 f. Ferrari, 1974 g. Chrysler, 1980 h. Audi, 1980 i. Chrysler, 1983	i. Ferrari, 1960 ii. Renault, 1985 iii. Ford, 2003 iv. Audi, 1976 v. Opel, 1983 vi. Ferrari, 1974 vii. Chrysler, 1980 viii. Audi, 1980 ix. Chrysler, 1983

And for unordered lists, available values are;

disc	circle	square
• Opel, 1980 • Chrysler, 1966 • Volvo, 1962 • Audi, 1990 • Ford, 1972 • Mercedes, 2003 • BMW, 1984 • Audi, 1975 • Volvo, 1973	○ Opel, 1980 ○ Chrysler, 1966 ○ Volvo, 1962 ○ Audi, 1990 ○ Ford, 1972 ○ Mercedes, 2003 ○ BMW, 1984 ○ Audi, 1975 ○ Volvo, 1973	▪ Opel, 1980 ▪ Chrysler, 1966 ▪ Volvo, 1962 ▪ Audi, 1990 ▪ Ford, 1972 ▪ Mercedes, 2003 ▪ BMW, 1984 ▪ Audi, 1975 ▪ Volvo, 1973

Definition Lists

Third type of dataList is definition lists that display inline description for each item, to use definition list set *type* option to *"definition"*. Detail content is provided with the facet called *"description"*.

```
<p:dataList value="#{carBean.cars}" var="car" type="definition">
      Model: #{car.model}, Year: #{car.year}
      <f:facet name="description">
            <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg"/>
      </f:facet>
</p:dataList>
```

Model: 61a10db5, Year: 2008
Model: 9efc3d27, Year: 1973
Model: 2d1a03f2, Year: 2009
Model: 987f2246, Year: 1963
Model: 14b594fc, Year: 1998

Ajax Pagination

DataList has a built-in paginator that is enabled by setting paginator option to true.

```
<p:dataList value="#{carBean.cars}" var="car" paginator="true" rows="10">
     #{car.manufacturer}, #{car.year}
</p:dataList>
```

Pagination configuration and usage is same as dataGrid, see pagination section in dataGrid documentation for more information and examples.

Selecting Data

Data selection can be implemented same as in dataGrid, see selecting data section in dataGrid documentation for more information and examples.

Client Side API

Widget: *PrimeFaces.widget.DataList*

Method	Params	Return Type	Description
getPaginator()	-	Paginator	Returns the paginator widget.

Skinning

DataList resides in a main div container which style and styleClass attributes apply. Following is the list of structural style classes;

Class	Applies
.ui-datalist	Main container element
.ui-datalist-content	Content container
.ui-datalist-data	Data container
.ui-datalist-item	Each item in list

As skinning style classes are global, see the main Skinning section for more information.

Tips

- DataList supports lazy loading data via LazyDataModel, see DataTable lazy loading section.
- If you need full control over list type markup, set type to "none". With this setting, datalist does not render item tags like li and behaves like ui:repeat.
- DataList provides two facets named *header* and *footer* that you can use to provide custom content at these locations.

3.26 DataTable

DataTable is an enhanced version of the standard Datatable that provides built-in solutions to many commons use cases like paging, sorting, selection, lazy loading, filtering and more.

Model	Year	Manufacturer	Color
62da385e	2000	Opel	Green
9429b69f	2007	Mercedes	Brown
0b1db0d0	2001	BMW	Orange
1edc33c9	1977	Audi	Red
c9a6048e	2009	Opel	Blue
272e6dc3	1964	Ford	Brown
cbde87b3	2007	Volkswagen	Yellow
c0f941e5	1998	Opel	White
41a1c297	2003	Volvo	Maroon

Info

Tag	**dataTable**
Component Class	**org.primefaces.component.datatable.DataTable**
Component Type	**org.primefaces.component.DataTable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DataTableRenderer**
Renderer Class	**org.primefaces.component.datatable.DataTableRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data to display.
var	null	String	Name of the request-scoped variable used to refer each data.
rows	null	Integer	Number of rows to display per page.
first	0	Integer	Index of the first row to be displayed
widgetVar	null	String	Name of the client side widget.

Name	Default	Type	Description
paginator	FALSE	boolean	Enables pagination.
paginatorTemplate	null	String	Template of the paginator.
rowsPerPageTemplate	null	String	Template of the rowsPerPage dropdown.
currentPageReportTemplate	null	String	Template of the currentPageReport UI.
pageLinks	10	Integer	Maximum number of page links to display.
paginatorPosition	both	String	Position of the paginator.
paginatorAlwaysVisible	TRUE	Boolean	Defines if paginator should be hidden if total data count is less than number of rows per page.
scrollable	FALSE	Boolean	Makes data scrollable with fixed header.
scrollHeight	null	Integer	Scroll viewport height.
scrollWidth	null	Integer	Scroll viewport width.
selectionMode	null	String	Enables row selection, valid values are "single" and "multiple".
selection	null	Object	Reference to the selection data.
rowIndexVar	null	String	Name of iterator to refer each row index.
emptyMessage	No records found.	String	Text to display when there is no data to display.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
dblClickSelect	FALSE	Boolean	Enables row selection on double click.
liveScroll	FALSE	Boolean	Enables live scrolling.
rowStyleClass	null	String	Style class for each row.
onExpandStart	null	String	Client side callback to execute before expansion.
resizableColumns	FALSE	Boolean	Enables column resizing.
sortBy	null	Object	Property to be used for default sorting.
sortOrder	ascending	String	"ascending" or "descending".
scrollRows	0	Integer	Number of rows to load on live scroll.
rowKey	null	String	Unique identifier of a row.
tableStyle	null	String	Inline style of the table element.
tableStyleClass	null	String	Style class of the table element.
filterEvent	keyup	String	Event to invoke filtering for input filters.

Name	Default	Type	Description
draggableColumns	FALSE	Boolean	Columns can be reordered with dragdrop when enabled.
editable	FALSE	Boolean	Controls incell editing.
lazy	FALSE	Boolean	Controls lazy loading.

Getting started with the DataTable

We will be using the same Car and CarBean classes described in DataGrid section.

```
<p:dataTable var="car" value="#{carBean.cars}">
      <p:column>
            <f:facet name="header">
                  <h:outputText value="Model" />
            </f:facet>
            <h:outputText value="#{car.model}" />
      </p:column>
      <p:column>
            <f:facet name="header">
                  <h:outputText value="Year" />
            </f:facet>
            <h:outputText value="#{car.year}" />
      </p:column>
      <p:column>
            <f:facet name="header">
                  <h:outputText value="Manufacturer" />
            </f:facet>
            <h:outputText value="#{car.manufacturer}" />
      </p:column>
      <p:column>
            <f:facet name="header">
                  <h:outputText value="Color" />
            </f:facet>
            <h:outputText value="#{car.color}" />
      </p:column>
</p:dataTable>
```

Header and Footer

Both datatable itself and columns can have headers and footers.

List of Cars

Model	Manufacturer	Color	Year
16c9b7c6	Mercedes	Maroon	1979
de0e4475	Volkswagen	Maroon	1994
d17a0cac	Ford	Black	1998
0db0095d	Ford	Red	1983
c09b2d08	Renault	Red	1962
a5e3c203	Volkswagen	Green	2007
196bd9e9	Ford	White	1994
111db4d2	Ford	Silver	1994
73b17bd0	Volvo	Blue	1973
8 digit code			1960-2010

In total there are 9 cars.

```
<p:dataTable var="car" value="#{carBean.cars}">

      <f:facet name="header">
            List of Cars
      </f:facet>

      <p:column>
            <f:facet name="header">
                  Model
            </f:facet>
                  #{car.model}
            <f:facet name="footer">
                  8 digit code
            </f:facet>
      </p:column>

      <p:column headerText="Year" footerText="1960-2010">
            #{car.year}
      </p:column>

      //more columns

      <f:facet name="header">
            In total there are #{fn:length(carBean.cars)} cars.
      </f:facet>

</p:dataTable>
```

headerText and *footerText* attributes on column are handy shortcuts for *header* and *footer* facets that just display plain texts.

Pagination

DataTable has a built-in ajax based paginator that is enabled by setting paginator option to true, see pagination section in dataGrid documentation for more information about customization.

```
<p:dataTable var="car" value="#{carBean.cars}" paginator="true" rows="10">
      //columns
 </p:dataTable>
```

Sorting

Defining *sortBy* attribute enables ajax based sorting on that particular column.

```
<p:dataTable var="car" value="#{carBean.cars}">
      <p:column sortBy="#{car.model}" headerText=”Model”>
            <h:outputText value="#{car.model}" />
      </p:column>
      ...more columns
</p:dataTable>
```

Instead of using the default sorting algorithm which uses a java comparator, you can plug-in your own sort method.

```
<p:dataTable var="car" value="#{carBean.cars}" dynamic="true">
      <p:column sortBy="#{car.model}" sortFunction="#{carBean.sortByModel}"
                  headerText=”Model”>
            <h:outputText value="#{car.model}" />
      </p:column>

      ...more columns
</p:dataTable>
```

```
public int sortByModel(Car car1, Car car2) {
      //return -1, 0 , 1 if car1 is less than, equal to or greater than car2
}
```

DataTable can display data sorted by default, to implement this use the *sortBy* option of datatable and optional *sortOrder*. Table below would be initially displayed as sorted by model.

```
<p:dataTable var="car" value="#{carBean.cars}" sortBy=”#{car.model}”>

      <p:column sortBy="#{car.model}" sortFunction="#{carBean.sortByModel}"
                  headerText=”Model”>
            <h:outputText value="#{car.model}" />
      </p:column>

      ...more columns
</p:dataTable>
```

Data Filtering

Similar to sorting, ajax based filtering is enabled at column level by setting *filterBy* option.

```
<p:dataTable var="car" value="#{carBean.cars}">

      <p:column filterBy="#{car.model}">
            <f:facet name="header">
                  <h:outputText value="Model" />
            </f:facet>
            <h:outputText value="#{car.model}" />
      </p:column>

      ...more columns

</p:dataTable>
```

Filtering is triggered with keyup event and filter inputs can be styled using *filterStyle* , *filterStyleClass* attributes. If you'd like to use a dropdown instead of an input field to only allow predefined filter values use *filterOptions* attribute and a collection/array of selectitems as value. In addition, *filterMatchMode* defines the built-in matcher which is *startsWith* by default. Following is an advanced filtering datatable with these options demonstrated.

```
<p:dataTable var="car" value="#{carBean.cars}" widgetVar="carsTable">

      <f:facet name="header">
            <p:outputPanel>
                  <h:outputText value="Search all fields:" />
                  <h:inputText id="globalFilter" onkeyup="carsTable.filter()" />
            </p:outputPanel>
      </f:facet>

    <p:column filterBy="#{car.model}" headerText="Model" filterMatchMode="contains">
      <h:outputText value="#{car.model}" />
    </p:column>

    <p:column filterBy="#{car.year}" headerText="Year" footerText="startsWith">
      <h:outputText value="#{car.year}" />
    </p:column>

    <p:column filterBy="#{car.manufacturer}" headerText="Manufacturer"
      filterOptions="#{carBean.manufacturerOptions}" filterMatchMode="exact">
      <h:outputText value="#{car.manufacturer}" />
    </p:column>

   <p:column filterBy="#{car.color}" headerText="Color" filterMatchMode="endsWith">
      <h:outputText value="#{car.color}" />
      </p:column>

</p:dataTable>
```

Search all fields:

Model	Year	Manufacturer	Color
		Select	
9f1e82ad	1989	Volkswagen	Black
c1362b1d	1968	Mercedes	Blue
ec1f0bb1	1962	Renault	Green
9b0b3fe3	2001	Mercedes	Yellow
de0517b3	2002	Volkswagen	Green
3e702116	1972	BMW	Blue
49612134	1994	Ford	Black
bf19778d	1983	Audi	Red
4ecd938b	1962	Opel	Yellow
contains	startsWith	exact	endsWith

Filter located at header is a global one applying on all fields, this is implemented by calling client side api method called *filter()*, important part is to specify the id of the input text as *globalFilter* which is a reserved identifier for datatable.

Row Selection

There are several ways to select row(s) from datatable. Let's begin by adding a Car reference for single selection and a Car array for multiple selection to the CarBean to hold the selected data.

```
public class CarBean {

      private List<Car> cars;

      private Car selectedCar;

      private Car[] selectedCars;

      public CarBean() {
            cars = new ArrayList<Car>();
            cars.add(new Car("myModel",2005,"ManufacturerX","blue"));
            //add more cars
      }

      //getters and setters
}
```

Single Selection with a Command Component

This method is implemented with a command component such as commandLink or commandButton. Selected row can be set to a server side instance by passing as a parameter if you are using EL 2.2 or using f:setPropertyActionListener.

```
<p:dataTable var="car" value="#{carBean.cars}">

      <p:column>
            <p:commandButton value="Select">
                  <f:setPropertyActionListener value="#{car}"
                        target="#{carBean.selectedCar}" />
            </p:commandButton>
      </p:column>

      ...columns

</p:dataTable>
```

Single Selection with Row Click

Previous method works when the button is clicked, if you'd like to enable selection wherever the row is clicked, use *selectionMode* option.

```
<p:dataTable var="car" value="#{carBean.cars}" selectionMode="single"
            selection="#{carBean.selectedCar}" rowKey="#{car.id}">
      ...columns
</p:dataTable>
```

Multiple Selection with Row Click

Multiple row selection is similar to single selection but selection should reference an array of the domain object displayed and user needs to use press modifier key(e.g. ctrl) during selection.

```
<p:dataTable var="car" value="#{carBean.cars}" selectionMode="multiple"
             selection="#{carBean.selectedCars}" rowKey="#{car.id}" >
      ...columns
</p:dataTable>
```

Selection on Double Click

By default, row based selection is enabled by click event, enable *dblClickSelect* so that clicking double on a row does the selection.

Single Selection with RadioButton

Selection a row with a radio button placed on each row is a common case, datatable has built-in support for this method so that you don't need to deal with h:selectOneRadio's and low level bits. In order to enable this feature, define a column with *selectionMode* set as single.

```
<p:dataTable var="car" value="#{carBean.cars}" selection="#{carBean.selecedCar}"
                   rowKey="#{car.id}">
      <p:column selectionMode="single"/>
      ...columns
</p:dataTable>
```

Multiple Selection with Checkboxes

Similar to how radio buttons are enabled, define a selection column with a multiple selectionMode. DataTable will also provide a selectAll checkbox at column header.

```
<p:dataTable var="car" value="#{carBean.cars}" selection="#{carBean.selecedCars}"
             rowKey="#{car.id}" >
      <p:column selectionMode="multiple"/>
      ...columns
</p:dataTable>
```

RowKey

RowKey should a unique identifier from your data model and used by datatable to find the selected rows. You can either define this key by using the rowKey attribute or by binding a data model which implements *org.primefaces.model.SelectableDataModel*.

Dynamic Columns

Dynamic columns is handy in case you can't know how many columns to render. Columns component is used to define the columns programmatically. It requires a collection as the value, two iterator variables called *var* and *columnIndexVar*. Features like *sortBy* and *filterBy* are supported however advanced features like editing is not.

```
<p:dataTable var="cars" value="#{tableBean.dynamicCars}" id="carsTable">
      <p:columns value="#{tableBean.columns}" var="column" columnIndexVar="colIndex">
            <f:facet name="header">
                  #{column}
            </f:facet>

            <h:outputText value="#{cars[colIndex].model}" /> <br />
            <h:outputText value="#{cars[colIndex].year}" /> <br />
            <h:outputText value="#{cars[colIndex].color}"/>
      </p:columns>
</p:dataTable>
```

```
public class CarBean {

      private List<String> columns;

      private List<Car[]> dynamicCars;

      public CarBean() {
            populateColumns();
            populateCars();
      }

      public void populateColumns() {
            columns = new ArrayList();

            for(int i = 0; i < 3; i++) {
                  columns.add("Brand:" + i);
            }

      }

      public void populateCars() {
            dynamicCars = new ArrayList<Car[]>();

            for(int i=0; i < 9; i++) {
                  Car[] cars = new Car[columns.size()];

                  for(int j = 0; j < columns.size(); j++) {
                        cars[j] = //Create car object
                  }
                  dynamicCars.add(cars);
            }
      }
}
```

Grouping

Grouping is defined by ColumnGroup component used to combine datatable header and footers.

```
<p:dataTable var="sale" value="#{carBean.sales}">

      <p:columnGroup type="header">
            <p:row>
                  <p:column rowspan="3" headerText="Manufacturer" />
                  <p:column colspan="4" headerText="Sales" />
            </p:row>
            <p:row>
                   <p:column colspan="2" headerText="Sales Count" />
                   <p:column colspan="2" headerText="Profit" />
            </p:row>
            <p:row>
                  <p:column headerText="Last Year" />
                  <p:column headerText="This Year" />
                  <p:column headerText="Last Year" />
                  <p:column headerText="This Year" />
            </p:row>
      </p:columnGroup>

      <p:column>
            #{sale.manufacturer}
      </p:column>
      <p:column>
            #{sale.lastYearProfit}%
      </p:column>
      <p:column>
            #{sale.thisYearProfit}%
      </p:column>
      <p:column>
            #{sale.lastYearSale}$
      </p:column>
      <p:column>
            #{sale.thisYearSale}$
      </p:column>
```

```
   <p:columnGroup type="footer">
      <p:row>
            <p:column colspan="3" style="text-align:right" footerText="Totals:"/>
            <p:column footerText="#{tableBean.lastYearTotal}$" />
            <p:column footerText="#{tableBean.thisYearTotal}$" />
      </p:row>
   </p:columnGroup>

</p:dataTable>
```

```
public class CarBean {

      private List<Manufacturer> sales;

      public CarBean() {
            sales = //create a list of BrandSale objects
      }

      public List<ManufacturerSale> getSales() {
            return this.sales;
      }
}
```

Manufacturer	Sales			
	Sales Count		Profit	
	Last Year	This Year	Last Year	This Year
Mercedes	90%	8%	28031$	25102$
BMW	14%	91%	18640$	28023$
Volvo	82%	24%	130$	77724$
Audi	7%	40%	2272$	33672$
Renault	10%	54%	98115$	40664$
Opel	63%	28%	10549$	93746$
Volkswagen	67%	38%	38242$	19063$
Chrysler	40%	63%	10146$	7697$
Ferrari	26%	70%	40384$	62298$
Ford	14%	94%	96052$	42233$
		Totals:	342561$	430222$

Scrolling

Scrolling is a way to display data with fixed header&footer, in order to enable scrolling set scrollable option to true, define a fixed height and/or width in pixels and set a fixed width to each column. **It is important to use width attribute of column instead of style attribute for scrollable case**.

```
<p:dataTable var="car" value="#{bean.data}" scrollable="true"scrollHeight="150">
      <p:column width="100" />
      //columns
</p:dataTable>
```

Model	Year	Manufacturer	Color
069794d7	1991	Volvo	Silver
4aeeec6c	1993	Ford	Green
09cbc05c	1983	Chrysler	Maroon
2d374a04	1964	Ferrari	Red
9c09bc54	1987	Volkswagen	Blue
25d45a08	1993	Opel	White
Model	Year	Year	Year

Simple scrolling renders all data to client and places a scrollbar, live scrolling is necessary to deal with huge data, in this case data is fetched whenever the scrollbar reaches bottom. Set *liveScroll* to enable this option;

```
<p:dataTable var="car" value="#{bean.data}" scrollable="true" scrollHeight="150"
      liveScroll="true">

      <p:column width="100" />
      //columns
</p:dataTable>
```

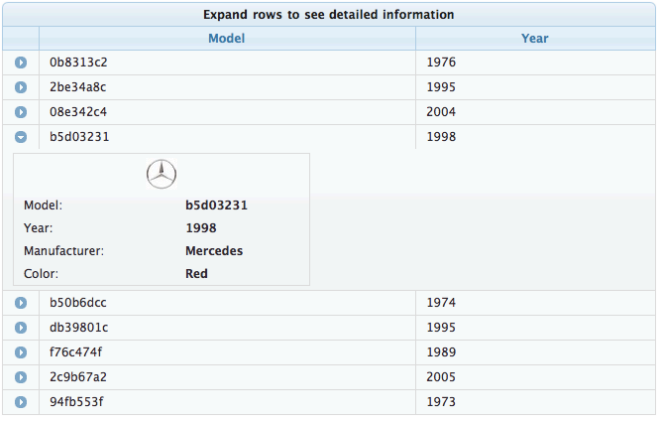
Scrolling has 3 modes; x, y and x-y scrolling that are defined by *scrollHeight* and *scrollWidth*.

Expandable Rows

RowToggler and *RowExpansion* facet are used to implement expandable rows.

```
<p:dataTable var="car" value="#{carBean.cars}">

      <f:facet name="header">
            Expand rows to see detailed information
      </f:facet>

      <p:column>
            <p:rowToggler />
      </p:column>

      //columns

      <p:rowExpansion>
            //Detailed content of a car
      </p:rowExpansion>

</p:dataTable>
```

p:rowToggler component places an expand/collapse icon, clicking on a collapsed row loads expanded content with ajax.



Incell Editing

Incell editing provides an easy way to display editable data. *p:cellEditor* is used to define the cell editor of a particular column and *p:rowEditor* is used to toggle edit/view displays of a row.

```
<p:dataTable var="car" value="#{carBean.cars}">

      <f:facet name="header">
           In-Cell Editing
       </f:facet>

      <p:column headerText="Model">

            <p:cellEditor>
               <f:facet name="output">
                   <h:outputText value="#{car.model}" />
               </f:facet>

               <f:facet name="input">
                   <h:inputText value="#{car.model}"/>
               </f:facet>

           </p:cellEditor>
      </p:column>

      //more columns with cell editors

      <p:column>
            <p:rowEditor />
      </p:column>

</p:dataTable>
```

In-Cell Editing

Model	Year	Manufacturer	Color	Options
824641ad	1976	Volvo	Yellow	
a9bf1625	1961	Volkswagen	Orange	
d859a7ba	1977	Ferrari	Brown	
9379f6f5	1961	Renault	Silver	
744a8017	1960	Chrysler	Silver	
80feefe5	2000	Opel	Yellow	
9e0c7267	1982	Opel	Red	
33124250	1984	Ford	Red	
0349899f	1977	Renault	Red	

When pencil icon is clicked, row is displayed in editable mode meaning input facets are displayed and output facets are hidden. Clicking tick icon only saves that particular row and cancel icon reverts the changes, both options are implemented with ajax.

See ajax behavior events section for more information about cell edit event.

Lazy Loading

Lazy Loading is a built-in feature of datatable to deal with huge datasets efficiently, regular ajax based pagination works by rendering only a particular page but still requires all data to be loaded into memory. Lazy loading datatable renders a particular page similarly but also only loads the page data into memory not the whole dataset. In order to implement this, you'd need to bind a *org.primefaces.model.LazyDataModel* as the value and implement *load* method and enable *lazy* option. Also you must implement *getRowData* and *getRowKey* if you have selection enabled.

```
<p:dataTable var="car" value="#{carBean.model}" paginator="true" rows="10"
      lazy="true">
      //columns
</p:dataTable>
```

```
public class CarBean {

      private LazyDataModel model;

      public CarBean() {
            model = new LazyDataModel() {

                  @Override
                  public void load(int first, int pageSize, String sortField,
                        SortOrder sortOrder, Map<String,String> filters) {

                        //load physical data
                  }
            };

            int totalRowCount = //logical row count based on a count query
            model.setRowCount(totalRowCount);
      }

      public LazyDataModel getModel() {
            return model;
      }
}
```

DataTable calls your load implementation whenever a paging, sorting or filtering occurs with following parameters;

- first: Offset of first data to start from
- pageSize: Number of data to load
- sortField: Name of sort field (e.g. "model" for sortBy="#{car.model}")
- sortOrder: SortOrder enum.
- filter: Filter map with field name as key (e.g. "model" for filterBy="#{car.model}") and value.

In addition to load method, totalRowCount needs to be provided so that paginator can display itself according to the logical number of rows to display.

SummaryRow

Summary is a helper component to display a summary for the grouping which is defined by the sortBy option.

```
<p:dataTable var="car" value="#{tableBean.cars}" sortBy="#{car.manufacturer}"
                  sortOrder="descending">

      <p:column headerText="Model" sortBy="#{car.model}">
            #{car.model}
      </p:column>

      <p:column headerText="Year" sortBy="#{car.year}">
            #{car.year}
      </p:column>

      <p:column headerText="Manufacturer" sortBy="#{car.manufacturer}">
            #{car.manufacturer}
      </p:column>

      <p:column headerText="Color" sortBy="#{car.color}">
            #{car.color}
      </p:column>

      <p:summaryRow>
            <p:column colspan="3" style="text-align:right">
                  Total:
            </p:column>

            <p:column>
                  #{tableBean.randomPrice}$
            </p:column>
      </p:summaryRow>
</p:dataTable>
```

Model	Year	Manufacturer	Color
30d423c1	1995	Volvo	Orange
caa74a90	2005	Volvo	White
2295d17b	1996	Volvo	Blue
d9548573	1990	Volvo	Black
3f2fddb1	1979	Volvo	Blue
c9cb10af	2007	Volvo	Maroon
d69007fb	1998	Volvo	Black
		Total:	**40272$**
986742ea	1966	Volkswagen	Orange
f5045e9a	2006	Volkswagen	Red
3498c563	1994	Volkswagen	Red
		Total:	**61413$**

SubTable

SubTable is a helper component to display nested collections. Example below displays a collection of players and a subtable for the stats collection of each player.

```
<p:dataTable var="player" value="#{tableBean.players}">

      <f:facet name="header">
            FCB Statistics
      </f:facet>

      <p:columnGroup type="header">
            <p:row>
              <p:column rowspan="2" headerText="Player" sortBy="#{player.name}"/>
              <p:column colspan="2" headerText="Stats" />
            </p:row>

            <p:row>
                  <p:column headerText="Goals" />
                  <p:column headerText="Assists" />
            </p:row>
      </p:columnGroup>

      <p:subTable var="stats" value="#{player.stats}">
            <f:facet name="header">
                  #{player.name}
            </f:facet>

            <p:column>
                  #{stats.season}
            </p:column>

            <p:column>
                  #{stats.goals}
            </p:column>

            <p:column>
                  #{stats.assists}
            </p:column>

            <p:columnGroup type="footer">
                  <p:row>
                        <p:column footerText="Totals: " style="text-align:right"/>
                        <p:column footerText="#{player.allGoals}" />
                        <p:column footerText="#{player.allAssists}" />
                  </p:row>
            </p:columnGroup>
      </p:subTable>

</p:dataTable>
```

FCB Statistics		
Player	Stats	
	Goals	Assists
Messi		
2005-2006	4	2
2006-2007	10	7
2007-2008	16	10
2008-2009	32	15
2009-2010	51	22
2010-2011	55	30
Totals:	168	86
Xavi		
2005-2006	6	15
2006-2007	10	20
2007-2008	12	22
2008-2009	9	24
2009-2010	8	21
2010-2011	10	25
Totals:	55	127
Iniesta		
2005-2006	4	12
2006-2007	7	9
2007-2008	10	14
2008-2009	15	17
2009-2010	14	16
2010-2011	17	22
Totals:	67	90

Ajax Behavior Events

DataTable provides many custom ajax behavior events for you to hook-in to various features.

Event	Listener Parameter	Fired
page	org.primefaces.event.data.PageEvent	On pagination.
sort	org.primefaces.event.data.SortEvent	When a column is sorted.
filter	org.primefaces.event.data.FilterEvent	On filtering.
rowSelect	org.primefaces.event.SelectEvent	When a row is being selected.
rowUnselect	org.primefaces.event.UnselectEvent	When a row is being unselected.
rowEdit	org.primefaces.event.RowEditEvent	When a row is edited.
colResize	org.primefaces.event.ColumnResizeEvent	When a column is being selected.
toggleSelect	-	When header checkbox is toggled.
colReorder	-	When columns are reordered.
rowSelectRadio	org.primefaces.event.SelectEvent	Row selection with radio.
rowSelectCheckbox	org.primefaces.event.SelectEvent	Row selection with checkbox.
rowUnselectcheckbox	org.primefaces.event.UnselectEvent	Row unselection with checkbox.
rowDblselect	org.primefaces.event.SelectEvent	Row selection with double click.

For example, datatable below makes an ajax request when a row is selected with a click on row.

```
<p:dataTable var="car" value="#{carBean.model}">
     <p:ajax event="rowSelect" update="another_component" />
     //columns
</p:dataTable>
```

Client Side API

Widget: *PrimeFaces.widget.DataTable*

Method	Params	Return Type	Description
getPaginator()	-	Paginator	Returns the paginator insance.
clearFilters()	-	void	Clears all column filters
getSelectedRowsCount()		Number	Returns number of selected rows.
selectRow(r, silent)	*r*: number or tr element as jQuery object, *silent*: flag to fire row select ajax behavior	void	Selects the given row.
unselectRow(r, silent)	*r*: number or tr element as jQuery object, *silent*: flag to fire row select ajax behavior	void	Unselects the given row.
unselectAllRows()	-	void	Unselects all rows.
toggleCheckAll()	-	void	Toggles header checkbox state.

Skinning

DataTable resides in a main container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Class	Applies
.ui-datatable	Main container element
.ui-datatable-data	Table body
.ui-datatable-empty-message	Empty message row
.ui-datatable-header	Table header
.ui-datatable-footer	Table footer
.ui-sortable-column	Sortable columns

Class	Applies
.ui-sortable-column-icon	Icon of a sortable icon
.ui-expanded-row-content	Content of an expanded row
.ui-row-toggler	Row toggler for row expansion
.ui-editable-column	Columns with a cell editor
.ui-cell-editor	Container of input and output controls of an editable cell
.ui-cell-editor-input	Container of input control of an editable cell
.ui-cell-editor-output	Container of output control of an editable cell
.ui-datatable-even	Even numbered rows
.ui-datatable-odd	Odd numbered rows

As skinning style classes are global, see the main Skinning section for more information.

3.27 DefaultCommand

Which command to submit the form with when enter key is pressed a common problem in web apps not just specific to JSF. Browsers tend to behave differently as there doesn't seem to be a standard and even if a standard exists, IE probably will not care about it. There are some ugly workarounds like placing a hidden button and writing javascript for every form in your app. DefaultCommand solves this problem by normalizing the command(e.g. button or link) to submit the form with on enter key press.

Info

Tag	**defaultCommand**
Component Class	**org.primefaces.component.defaultcommand.DefaultCommand**
Component Type	**org.primefaces.component.DefaultCommand**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DefaultCommandRenderer**
Renderer Class	**org.primefaces.component.defaultcommand.DefaultCommandRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
target	null	String	Identifier of the default command component.
context	null	String	Identifier of the grouping component to enable multiple default commands in a form.

Getting Started with the DefaultCommand

DefaultCommand must be nested inside a form requires *target* option to reference a clickable command. Example below triggers *btn2* when enter key is pressed. Note that an input must have focused due to browser nature.

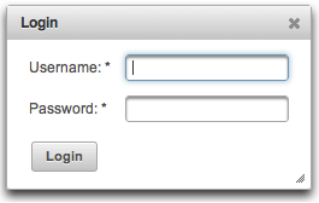
```
<h:form id="form">

      <h:panelGrid columns="3" cellpadding="5">
            <h:outputLabel for="name" value="Name:" style="font-weight:bold"/>
            <p:inputText id="name" value="#{defaultCommandBean.text}" />
            <h:outputText value="#{defaultCommandBean.text}" id="display" />
      </h:panelGrid>

      <p:commandButton value="Button1" id="btn1" actionListener="#{bean.submit1}"
                  ajax="false"/>
      <p:commandButton value="Button2" id="btn2" actionListener="#{bean.submit2}" />

      <h:commandButton value="Button3" id="btn3" actionListener="#{bean.submit3}" />

      <p:defaultCommand target="bt2" />

</h:form>
```

Tips

- If you need multiple default commands on same page use context attribute that refers to the ancestor component of the target input.

3.28 Dialog

Dialog is a panel component that can overlay other elements on page.



Info

Tag	**dialog**
Component Class	**org.primefaces.component.dialog.Dialog**
Component Type	**org.primefaces.component.Dialog**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DialogRenderer**
Renderer Class	**org.primefaces.component.dialog.DialogRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
header	null	String	Text of the header
draggable	TRUE	Boolean	Specifies draggability
resizable	TRUE	Boolean	Specifies resizability
modal	FALSE	Boolean	Enables modality.
visible	FALSE	Boolean	When enabled, dialog is visible by default.
width	auto	Integer	Width of the dialog
height	auto	Integer	Height of the dialog
minWidth	150	Integer	Minimum width of a resizable dialog.

Name	Default	Type	Description
minHeight	0	Integer	Minimum height of a resizable dialog.
style	null	String	Inline style of the dialog.
styleClass	null	String	Style class of the dialog
showEffect	null	String	Effect to use when showing the dialog
hideEffect	null	String	Effect to use when hiding the dialog
position	null	String	Defines where the dialog should be displayed
closable	TRUE	Boolean	Defines if close icon should be displayed or not
onShow	null	String	Client side callback to execute when dialog is displayed.
onHide	null	String	Client side callback to execute when dialog is hidden.
appendToBody	FALSE	Boolean	Appends dialog as a child of document body.
showHeader	TRUE	Boolean	Defines visibility of the header content.
footer	null	String	Text of the footer.
dynamic	FALSE	Boolean	Enables lazy loading of the content with ajax.
minimizable	FALSE	Boolean	Whether a dialog is minimizable or not.
maximizable	FALSE	Boolean	Whether a dialog is maximizable or not.

Getting started with the Dialog

Dialog is a panel component containing other components, note that by default dialog is not visible.

```
 <p:dialog>
      <h:outputText value="Resistance to PrimeFaces is Futile!" />
      //Other content
</p:dialog>
```

Show and Hide

Showing and hiding the dialog is easy using the client side api.

```
 <p:dialog header="Header Text" widgetVar="dlg">
      //Content
</p:dialog>

<p:commandButton value="Show" type="button" onclick="dlg.show()" />
<p:commandButton value="Hide" type="button" onclick="dlg.hide()" />
```

Effects

There are various effect options to be used when displaying and closing the dialog. Use *showEffect* and *hideEffect* options to apply these effects;

- blind
- bounce
- clip
- drop
- explode
- fade
- fold
- highlight
- puff
- pulsate
- scale
- shake
- size
- slide
- transfer

```
<p:dialog showEffect="fade" hideEffect="explode" ...>
     //...
</p:dialog>
```

Position

By default dialog is positioned at center of the viewport and *position* option is used to change the location of the dialog. Possible values are;

- Single string value like *'center', 'left', 'right', 'top', 'bottom'* representing the position within viewport.
- Comma separated x and y coordinate values like *200, 500*
- Comma separated position values like *'top','right'*. (Use single quotes when using a combination)

Some examples are described below;

```
<p:dialog position="top" ...>
```

```
<p:dialog position="left,top" ...>
```

```
<p:dialog position="200,50" ...>
```

Ajax Behavior Events

close event is one of the ajax behavior events provided by dialog that is fired when the dialog is hidden. If there is a listener defined it'll be executed by passing an instance of *org.primefaces.event.CloseEvent*.

Example below adds a FacesMessage when dialog is closed and updates the messages component to display the added message.

```
<p:dialog>
      <p:ajax event="close" listener="#{dialogBean.handleClose}" update="msg" />
      //Content
</p:dialog>

<p:messages id="msg" />
```

```
public class DialogBean {

      public void handleClose(CloseEvent event) {
            //Add facesmessage
      }
}
```

Two other ajax behavior events are **maximize** and **minimize** that are invoked when dialog is maximized or minimized.

Client Side Callbacks

Similar to close listener, onShow and onHide are handy callbacks for client side in case you need to execute custom javascript.

```
<p:dialog onShow="alert('Visible')" onHide="alert('Hidden')">
      //Content
</p:dialog>
```

Client Side API

Widget: *PrimeFaces.widget.Dialog*

Method	Params	Return Type	Description
show()	-	void	Displays dialog.
hide()	-	void	Closes dialog.

Skinning

Dialog resides in a main container element which *styleClass* option apply. Following is the list of structural style classes;

Style Class	Applies
.ui-dialog	Container element of dialog
.ui-dialog-titlebar	Title bar
.ui-dialog-title-dialog	Header text
.ui-dialog-titlebar-close	Close icon
.ui-dialog-content	Dialog body

As skinning style classes are global, see the main Skinning section for more information.

Tips

- Use appendToBody with care as the page definition and html dom would be different, for example if dialog is inside an h:form component and appendToBody is enabled, on the browser dialog would be outside of form and may cause unexpected results. In this case, nest a form inside a dialog.
- Do not place dialog inside tables, containers likes divs with relative positioning or with non-visible overflow defined, in cases like these functionality might be broken. This is not a limitation but a result of DOM model. For example dialog inside a layout unit, tabview, accordion are a couple of examples. Same applies to confirmDialog as well.

3.29 Drag&Drop

Drag&Drop utilities of PrimeFaces consists of two components; Draggable and Droppable.

3.29.1 Draggable

Info

Tag	**draggable**
Component Class	**org.primefaces.component.dnd.Draggable**
Component Type	**org.primefaces.component.Draggable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DraggableRenderer**
Renderer Class	**org.primefaces.component.dnd.DraggableRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
proxy	FALSE	Boolean	Displays a proxy element instead of actual element.
dragOnly	FALSE	Boolean	Specifies a draggable that can't be dropped.
for	null	String	Id of the component to add draggable behavior
disabled	FALSE	Boolean	Disables draggable behavior when true.
axis	null	String	Specifies drag axis, valid values are 'x' and 'y'.
containment	null	String	Constraints dragging within the boundaries of containment element
helper	null	String	Helper element to display when dragging
revert	FALSE	Boolean	Reverts draggable to it's original position when not dropped onto a valid droppable
snap	FALSE	Boolean	Draggable will snap to edge of near elements

Name	Default	Type	Description
snapMode	null	String	Specifies the snap mode. Valid values are 'both', 'inner' and 'outer'.
snapTolerance	20	Integer	Distance from the snap element in pixels to trigger snap.
zindex	null	Integer	ZIndex to apply during dragging.
handle	null	String	Specifies a handle for dragging.
opacity	1	Double	Defines the opacity of the helper during dragging.
stack	null	String	In stack mode, draggable overlap is controlled automatically using the provided selector, dragged item always overlays other draggables.
grid	null	String	Dragging happens in every x and y pixels.
scope	null	String	Scope key to match draggables and droppables.
cursor	crosshair	String	CSS cursor to display in dragging.
dashboard	null	String	Id of the dashboard to connect with.

Getting started with Draggable

Any component can be enhanced with draggable behavior, basically this is achieved by defining the id of component using the *for* attribute of draggable.

```
<p:panel id="pnl" header="Draggable Panel">
      <h:outputText value="This is actually a regular panel" />
</p:panel>

<p:draggable for="pnl"/>
```

If you omit the for attribute, parent component will be selected as the draggable target.

```
<h:graphicImage id="campnou" value="/images/campnou.jpg">
      <p:draggable />
</h:graphicImage>
```

Handle

By default any point in dragged component can be used as handle, if you need a specific handle, you can define it with handle option. Following panel is dragged using it's header only.

```
<p:panel id="pnl" header="Draggable Panel">
      <h:outputText value="I can only be dragged using my header" />
</p:panel>
<p:draggable for="pnl" handle="div.ui-panel-titlebar"/>
```

Drag Axis

Dragging can be limited to either horizontally or vertically.

```
<p:panel id="pnl" header="Draggable Panel">
      <h:outputText value="I am dragged on an axis only" />
</p:panel>

<p:draggable for="pnl" axis="x or y"/>
```

Clone

By default, actual component is used as the drag indicator, if you need to keep the component at it's original location, use a clone helper.

```
<p:panel id="pnl" header="Draggable Panel">
      <h:outputText value="I am cloned" />
</p:panel>

<p:draggable for="pnl" helper="clone"/>
```

Revert

When a draggable is not dropped onto a matching droppable, revert option enables the component to move back to it's original position with an animation.

```
<p:panel id="pnl" header="Draggable Panel">
   <h:outputText value="I will be reverted back to my original position" />
</p:panel>

<p:draggable for="pnl" revert="true"/>
```

Opacity

During dragging, opacity option can be used to give visual feedback, helper of following panel's opacity is reduced in dragging.

```
<p:panel id="pnl" header="Draggable Panel">
   <h:outputText value="My opacity is lower during dragging" />
</p:panel>

<p:draggable for="pnl" opacity="0.5"/>
```

Grid

Defining a grid enables dragging in specific pixels. This value takes a comma separated dimensions in x,y format.

```
<p:panel id="pnl" header="Draggable Panel">
   <h:outputText value="I am dragged in grid mode" />
</p:panel>

<p:draggable for="pnl" grid="20,40"/>
```

Containment

A draggable can be restricted to a certain section on page, following draggable cannot go outside of it's parent.

```
<p:outputPanel layout="block" style="width:400px;height:200px;">
      <p:panel id="conpnl" header="Restricted">
         <h:outputText value="I am restricted to my parent's boundaries" />
      </p:panel>
</p:outputPanel>

<p:draggable for="conpnl" containment="parent" />
```

3.29.2 Droppable

Info

Tag	**droppable**
Component Class	**org.primefaces.component.dnd.Droppable**
Component Type	**org.primefaces.component.Droppable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DroppableRenderer**
Renderer Class	**org.primefaces.component.dnd.DroppableRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Variable name of the client side widget
for	null	String	Id of the component to add droppable behavior
disabled	FALSE	Boolean	Disables of enables droppable behavior
hoverStyleClass	null	String	Style class to apply when an acceptable draggable is dragged over.
activeStyleClass	null	String	Style class to apply when an acceptable draggable is being dragged.
onDrop	null	String	Client side callback to execute when a draggable is dropped.
accept	null	String	Selector to define the accepted draggables.
scope	null	String	Scope key to match draggables and droppables.
tolerance	null	String	Specifies the intersection mode to accept a draggable.
datasource	null	String	Id of a UIData component to connect with.

Getting Started with Droppable

Usage of droppable is very similar to draggable, droppable behavior can be added to any component specified with the for attribute.

```
<p:outputPanel id="slot" styleClass="slot" />

<p:droppable for="slot" />
```

slot styleClass represents a small rectangle.

```
<style type="text/css">
      .slot {
            background:#FF9900;
            width:64px;
            height:96px;
            display:block;
      }
</style>
```

If *for* attribute is omitted, parent component becomes droppable.

```
<p:outputPanel id="slot" styleClass="slot">
      <p:droppable />
</p:outputPanel>
```

Ajax Behavior Events

drop is the only and default ajax behavior event provided by droppable that is processed when a valid draggable is dropped. In case you define a listener it'll be executed by passing an *org.primefaces.event.DragDrop* event instance parameter holding information about the dragged and dropped components.

Following example shows how to enable draggable images to be dropped on droppables.

```
<p:graphicImage id="messi" value="barca/messi_thumb.jpg" />
<p:draggable for="messi"/>

<p:outputPanel id="zone" styleClass="slot" />
<p:droppable for="zone">
      <p:ajax listener="#{ddController.onDrop}" />
</p:droppable>
```

```
public void onDrop(DragDropEvent ddEvent) {
      String draggedId = ddEvent.getDragId();       //Client id of dragged component
      String droppedId = ddEvent.getDropId();       //Client id of dropped component
      Object data = ddEvent.getData();              //Model object of a datasource
}
```

onDrop

onDrop is a client side callback that is invoked when a draggable is dropped, it gets two parameters event and ui object holding information about the drag drop event.

```
<p:outputPanel id="zone" styleClass="slot" />
<p:droppable for="zone" onDrop="handleDrop"/>
```

```
function handleDrop(event, ui) {
      var draggable = ui.draggable,    //draggable element, a jQuery object
      helper = ui.helper,              //helper element of draggable, a jQuery object
      position = ui.position,          //position of draggable helper
      offset = ui.offset;              //absolute position of draggable helper

}
```

DataSource

Droppable has special care for data elements that extend from UIData(e.g. datatable, datagrid), in order to connect a droppable to accept data from a data component define datasource option as the id of the data component. Example below show how to drag data from datagrid and drop onto a droppable to implement a dragdrop based selection. Dropped cars are displayed with a datatable.

```
public class TableBean {

      private List<Car> availableCars;
      private List<Car> droppedCars;

      public TableBean() {
            availableCars = //populate data
      }

      //getters and setters

      public void onCarDrop(DragDropEvent event) {
            Car car = ((Car) ddEvent.getData());
            droppedCars.add(car);
            availableCars.remove(car);
      }
}
```

```
<h:form id="carForm">
 <p:fieldset legend="AvailableCars">
   <p:dataGrid id="availableCars" var="car"
                  value="#{tableBean.availableCars}" columns="3">
     <p:column>
       <p:panel id="pnl" header="#{car.model}" style="text-align:center">
             <p:graphicImage value="/images/cars/#{car.manufacturer}.jpg" />
       </p:panel>
           <p:draggable for="pnl" revert="true" h
                        andle=".ui-panel-titlebar" stack=".ui-panel"/>
     </p:column>
   </p:dataGrid>
 </p:fieldset>

 <p:fieldset id="selectedCars" legend="Selected Cars" style="margin-top:20px">
     <p:outputPanel id="dropArea">

       <h:outputText value="!!!Drop here!!!"
           rendered="#{empty tableBean.droppedCars}" style="font-size:24px;" />

       <p:dataTable var="car" value="#{tableBean.droppedCars}"
           rendered="#{not empty tableBean.droppedCars}">
           <p:column headerText="Model">
                 <h:outputText value="#{car.model}" />
           </p:column>
           <p:column headerText="Year">
                 <h:outputText value="#{car.year}" />
            </p:column>
            <p:column headerText="Manufacturer">
                  <h:outputText value="#{car.manufacturer}" />
            </p:column>
            <p:column headerText="Color">
                 <h:outputText value="#{car.color}" />
            </p:column>
        </p:dataTable>
  </p:outputPanel>
</p:fieldset>

<p:droppable for="selectedCars" tolerance="touch"
      activeStyleClass="ui-state-highlight" datasource="availableCars"
      onDrop="handleDrop"/>
      <p:ajax listener="#{tableBean.onCarDrop}" update="dropArea availableCars" />
</p:droppable>

</h:form>

<script type="text/javascript">
      function handleDrop(event, ui) {
            ui.draggable.fadeOut('fast');          //fade out the dropped item
      }
</script>
```

Tolerance

There are four different tolerance modes that define the way of accepting a draggable.

Mode	Description
fit	draggable should overlap the droppable entirely
intersect	draggable should overlap the droppable at least 50%
pointer	pointer of mouse should overlap the droppable
touch	droppable should overlap the droppable at any amount

Acceptance

You can limit which draggables can be dropped onto droppables using scope attribute which a draggable also has. Following example has two images, only first image can be accepted by droppable.

```
<p:graphicImage id="messi" value="barca/messi_thumb.jpg" />
<p:draggable for="messi" scope="forward"/>

<p:graphicImage id="xavi" value="barca/xavi_thumb.jpg" />
<p:draggable for="xavi" scope="midfield"/>

<p:outputPanel id="forwardsonly" styleClass="slot" scope="forward" />
<p:droppable for="forwardsonly" />
```

Skinning

hoverStyleClass and *activeStyleClass* attributes are used to change the style of the droppable when interacting with a draggable.

3.30 Dock

Dock component mimics the well known dock interface of Mac OS X.

Info

Tag	**dock**
Component Class	**org.primefaces.component.dock.Dock**
Component Type	**org.primefaces.component.Dock**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.DockRenderer**
Renderer Class	**org.primefaces.component.dock.DockRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
model	null	MenuModel	MenuModel instance to create menus programmatically
position	bottom	String	Position of the dock, *bottom* or *top*.
itemWidth	40	Integer	Initial width of items.
maxWidth	50	Integer	Maximum width of items.
proximity	90	Integer	Distance to enlarge.
halign	center	String	Horizontal alignment,
widgetVar	null	String	Name of the client side widget.

Getting started with the Dock

A dock is composed of menuitems.

```
<p:dock>
      <p:menuitem value="Home" icon="/images/dock/home.png" url="#" />
      <p:menuitem value="Music" icon="/images/dock/music.png" url="#" />
      <p:menuitem value="Video" icon="/images/dock/video.png" url="#"/>
      <p:menuitem value="Email" icon="/images/dock/email.png" url="#"/>
      <p:menuitem value="Link" icon="/images/dock/link.png" url="#"/>
      <p:menuitem value="RSS" icon="/images/dock/rss.png" url="#"/>
      <p:menuitem value="History" icon="/images/dock/history.png" url="#"/>
</p:dock>
```

Position

Dock can be located in two locations, *top* or *bottom* (default). For a dock positioned at top set position to top.

Dock Effect

When mouse is over the dock items, icons are zoomed in. The configuration of this effect is done via the maxWidth and proximity attributes.

Dynamic Menus

Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

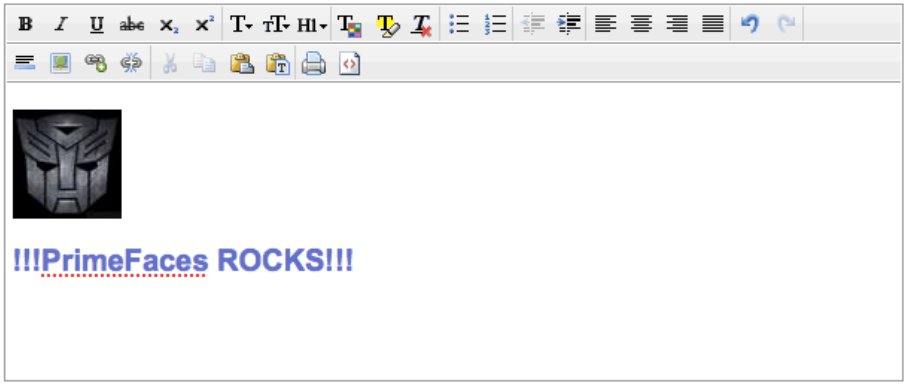
Following is the list of structural style classes, {positon} can be *top* or *bottom*.

Style Class	Applies
.ui-dock-{position}	Main container.
.ui-dock-container-{position}	Menu item container.
.ui-dock-item-{position}	Each menu item.

As skinning style classes are global, see the main Skinning section for more information.

3.31 Editor

Editor is an input component with rich text editing capabilities.



Info

Tag	**editor**
Component Class	**org.primefaces.component.editor.Editor**
Component Type	**org.primefaces.component.Editor**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.EditorRenderer**
Renderer Class	**org.primefaces.component.editor.EditorRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.

Name	Default	Type	Description
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required.
validator	null	MethodExpr	A method expression that refers to a method validationg the input.
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fails.
widgetVar	null	String	Name of the client side widget.
controls	null	String	List of controls to customize toolbar.
height	null	Integer	Height of the editor.
width	null	Integer	Width of the editor.
disabled	FALSE	Boolean	Disables editor.
style	null	String	Inline style of the editor container.
styleClass	null	String	Style class of the editor container.
onchange	null	String	Client side callback to execute when editor data changes.

Getting started with the Editor

Rich Text entered using the Editor is passed to the server using *value* expression.

```
public class Bean {

      private String text;

      //getter and setter
```

```
<p:editor value="#{bean.text}" />
```

Custom Toolbar

Toolbar of editor is easy to customize using *controls* option;

```
<p:editor value="#{bean.text}" controls="bold italic underline strikethrough" />
```

Here is the full list of all available controls;

• bold • italic • underline • strikethrough • subscript • superscript • font • size • style • color • highlight • bullets • numbering • alignleft • center • alignright	• justify • undo • redo • rule • image • link • unlink • cut • copy • paste • pastetext • print • source • outdent • indent • removeFormat

Client Side API

Widget: *PrimeFaces.widget.Editor*

Method	Params	Return Type	Description
init()	-	void	Initializes a lazy editor, subsequent calls do not reinit the editor.
saveHTML()	-	void	Saves html text in iframe back to the textarea.
clear()	-	void	Clears the text in editor.

Method	Params	Return Type	Description
enable()	-	void	Enables editing.
disable()	-	void	Disables editing.
focus()	-	void	Adds cursor focus to edit area.
selectAll()	-	void	Selects all text in editor.
getSelectedHTML()	-	String	Returns selected text as HTML.
getSelectedText()	-	String	Returns selected text in plain format.

Skinning

Following is the list of structural style classes.

Style Class	Applies
.ui-editor	Main container.
.ui-editor-toolbar	Toolbar of editor.
.ui-editor-group	Button groups.
.ui-editor-button	Each button.
.ui-editor-divider	Divider to separate buttons.
.ui-editor-disabled	Disabled editor controls.
.ui-editor-list	Dropdown lists.
.ui-editor-color	Color picker.
.ui-editor-popup	Popup overlays.
.ui-editor-prompt	Overlays to provide input.
.ui-editor-message	Overlays displaying a message.

Editor is not integrated with ThemeRoller as there is only one icon set for the controls.

3.32 Effect

Effect component is based on the jQuery UI effects library.

Info

Tag	**effect**
Tag Class	**org.primefaces.component.effect.EffectTag**
Component Class	**org.primefaces.component.effect.Effect**
Component Type	**org.primefaces.component.Effect**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.EffectRenderer**
Renderer Class	**org.primefaces.component.effect.EffectRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
effect	null	String	Name of the client side widget.
event	null	String	Dom event to attach the event that executes the animation
type	null	String	Specifies the name of the animation
for	null	String	Component that is animated
speed	1000	Integer	Speed of the animation in ms
delay	null	Integer	Time to wait until running the effect.

Getting started with Effect

Effect component needs a trigger and target which is effect's parent by default. In example below clicking outputText (trigger) will run the pulsate effect on outputText(target) itself.

```
<h:outputText value="#{bean.value}">
  <p:effect type="pulsate" event="click" />
</h:outputText>
```

Effect Target

There may be cases where you want to display an effect on another target on the same page while keeping the parent as the trigger. Use *for* option to specify a target.

```
<h:outputLink id="lnk" value="#">
      <h:outputText value="Show the Barca Temple" />
      <p:effect type="appear" event="click" for="img" />
</h:outputLink>

<p:graphicImage id="img" value="/ui/barca/campnou.jpg" style="display:none"/>
```

With this setting, outputLink becomes the trigger for the effect on graphicImage. When the link is clicked, initially hidden graphicImage comes up with a fade effect.

Note: It's important for components that have the effect component as a child to have an assigned id because some components do not render their clientId's if you don't give them an id explicitly.

List of Effects

Following is the list of effects supported by PrimeFaces.

- blind
- clip
- drop
- explode
- fold
- puff
- slide
- scale
- bounce
- highlight
- pulsate
- shake
- size
- transfer

Effect Configuration

Each effect has different parameters for animation like colors, duration and more. In order to change the configuration of the animation, provide these parameters with the f:param tag.

```
<h:outputText value="#{bean.value}">
  <p:effect type="scale" event="mouseover">
     <f:param name="percent" value="90"/>
  </p:effect>
</h:outputText>
```

It's important to provide string options with single quotes.

```
<h:outputText value="#{bean.value}">
  <p:effect type="blind" event="click">
     <f:param name="direction" value="'horizontal'" />
   </p:effect>
</h:outputText>
```

For the full list of configuration parameters for each effect, please see the jquery documentation;

http://docs.jquery.com/UI/Effects

Effect on Load

Effects can also be applied to any JSF component when page is loaded for the first time or after an ajax request is completed by using *load* as the event name. Following example animates messages with pulsate effect after ajax request completes.

```
<p:messages id="messages">
      <p:effect type="pulsate" event="load" delay=”500”>
            <f:param name="mode" value="'show'" />
      </p:effect>
</p:messages>

<p:commandButton value="Save" actionListener="#{bean.action}" update="messages"/>
```

3.33 FeedReader

FeedReader is used to display content from a feed.

Info

Tag	**feedReader**
Component Class	**org.primefaces.component.feedreader.FeedReader**
Component Type	**org.primefaces.component.FeedReader**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.FeedReaderRenderer**
Renderer Class	**org.primefaces.component.feedreader.FeedReaderRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	URL of the feed.
var	null	String	Iterator to refer each item in feed.
size	Unlimited	Integer	Number of items to display.

Getting started with FeedReader

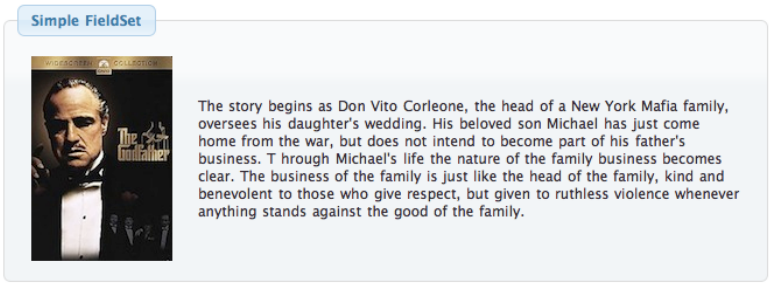
FeedReader requires a feed url to display and renders it's content for each feed item.

```
<p:feedReader value="http://rss.news.yahoo.com/rss/sports" var="feed">
      <h:outputText value="#{feed.title}" style="font-weight: bold"/>
      <h:outputText value="#{feed.description.value}" escape="false"/>
      <p:separator />
      <f:facet name="error">
            Something went wrong.
      </f:facet>
</p:feedReader>
```

Note that you need the ROME library in your classpath to make feedreader work.

3.34 Fieldset

Fieldset is a grouping component as an extension to html fieldset.


Simple FieldSet
The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. T hrough Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.

Info

Tag	**fieldset**
Component Class	**org.primefaces.component.fieldset.Fieldset**
Component Type	**org.primefaces.component.Fieldset**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.FieldsetRenderer**
Renderer Class	**org.primefaces.component.fieldset.FieldsetRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
legend	null	String	Title text.
style	null	String	Inline style of the fieldset.
styleClass	null	String	Style class of the fieldset.
toggleable	FALSE	Boolean	Makes content toggleable with animation.
toggleSpeed	500	Integer	Toggle duration in milliseconds.
collapsed	FALSE	Boolean	Defines initial visibility state of content.

Getting started with Fieldset

Fieldset is used as a container component for its children.

```
<p:fieldset legend="Simple Fieldset">
      <h:panelGrid column="2">
            <p:graphicImage value="/images/godfather/1.jpg" />
            <h:outputText value="The story begins as Don Vito Corleone ..." />
      </h:panelGrid>
</p:fieldset>
```

Legend

Legend can be defined in two ways, with legend attribute as in example above or using legend facet. Use facet way if you need to place custom html other than simple text.

```
<p:fieldset>
      <f:facet name="legend">
      </f:facet>

      //content
</p:fieldset>
```

When both legend attribute and legend facet are present, facet is chosen.

Toggleable Content

Clicking on fieldset legend can toggle contents, this is handy to use space efficiently in a layout. Set toggleable to true to enable this feature.

```
<p:fieldset legend="Toggleable Fieldset" toggleable="true">
      <h:panelGrid column="2">
            <p:graphicImage value="/images/godfather/2.jpg" />
            <h:outputText value="Francis Ford Coppolas' legendary ..." />
      </h:panelGrid>
</p:fieldset>
```

Toggleable Fieldset
Francis Ford Coppola's legendary continuation and sequel to his landmark 1972 film, The_Godfather, parallels the young Vito Corleone's rise with his son Michael's spiritual fall, deepening The_Godfather's depiction of the dark side of the American dream. In the early 1900s, the child Vito flees his Sicilian village for America after the local Mafia kills his family. Vito struggles to make a living, legally or illegally, for his wife and growing brood in Little Italy, killing the local Black Hand Fanucci after he demands his customary cut of the tyro's business. With Fanucci gone, Vito's communal stature grows.

Ajax Behavior Events

toggle is the default and only ajax behavior event provided by fieldset that is processed when the content is toggled. In case you have a listener defined, it will be invoked by passing an instance of *org.primefaces.event.ToggleEvent.*

Here is an example that adds a facesmessage and updates growl component when fieldset is toggled.

```
<p:growld id="messages" />

<p:fieldset legend="Toggleable Fieldset" toggleable="true"
      <p:ajax listener="#{bean.onToggle}" update="messages">

      //content

</p:fieldset>
```

```
public void onToggle(ToggleEvent event) {
      Visibility visibility = event.getVisibility();
      FacesMessage msg = new FacesMessage();
      msg.setSummary("Fieldset " + event.getId() + " toggled");
      msg.setDetail("Visibility: " + visibility);

      FacesContext.getCurrentInstance().addMessage(null, msg);
}
```

Client Side API

Widget: *PrimeFaces.widget.Fieldset*

Method	Params	Return Type	Description
toggle()	-	void	Toggles fieldset content.

Skinning

style and *styleClass* options apply to the fieldset.

Following is the list of structural style classes;

Style Class	Applies
.ui-fieldset	Main container
.ui-fieldset-toggleable	Main container when fieldset is toggleable
.ui-fieldset .ui-fieldset-legend	Legend of fieldset

Style Class	Applies
.ui-fieldset-toggleable .ui-fieldset-legend	Legend of fieldset when fieldset is toggleable
.ui-fieldset .ui-fieldset-toggler	Toggle icon on fieldset

As skinning style classes are global, see the main Skinning section for more information.

Tips

- A collapsed fieldset will remain collapsed after a postback since fieldset keeps its toggle state internally, you don't need to manage this using toggleListener and collapsed option.

3.35 FileDownload

The legacy way to present dynamic binary data to the client is to write a servlet or a filter and stream the binary data. FileDownload presents an easier way to do the same.

Info

Tag	**fileDownload**
ActionListener Class	**org.primefaces.component.filedownload.FileDownloadActionListener**

Attributes

Name	Default	Type	Description
value	null	StreamedContent	A streamed content instance
contextDisposition	attachment	String	Specifies display mode.

Getting started with FileDownload

A user command action is required to trigger the filedownload process. FileDownload can be attached to any command component like a commandButton or commandLink.

The value of the FileDownload must be an *org.primefaces.model.StreamedContent* instance. We suggest using the built-in *DefaultStreamedContent* implementation. First parameter of the constructor is the binary stream, second is the mimeType and the third parameter is the name of the file.

```
public class FileBean {

    private StreamedContent file;

    public FileDownloadController() {
        InputStream stream = this.getClass().getResourceAsStream("yourfile.pdf");
        file = new DefaultStreamedContent(stream, "application/pdf",
                            "downloaded_file.pdf");
      }

      public StreamedContent getFile() {
            return this.file;

      }
}
```

This streamed content should be bound to the value of the fileDownload.

```
<h:commandButton value="Download">
      <p:fileDownload value="#{fileBean.file}" />
</h:commandButton>
```

Similarly a more graphical presentation would be to use a commandlink with an image.

```
<h:commandLink value="Download">
      <p:fileDownload value="#{fileBean.file}"/>
      <h:graphicImage value="pdficon.gif" />
</h:commandLink>
```

If you'd like to use PrimeFaces commandButton and commandLink, disable ajax option as fileDownload requires a full page refresh to present the file.

```
<p:commandButton value="Download" ajax="false">
      <p:fileDownload value="#{fileBean.file}" />
</p:commandButton>
```

```
<p:commandLink value="Download" ajax="false">
      <p:fileDownload value="#{fileBean.file}"/>
      <h:graphicImage value="pdficon.gif" />
</p:commandLink>
```

ContentDisposition

By default, content is displayed as an *attachment* with a download dialog box, another alternative is the *inline* mode, in this case browser will try to open the file internally without a prompt. Note that content disposition is not part of the http standard although it is widely implemented.

Monitor Status

As fileDownload process is non-ajax, ajaxStatus cannot apply. Still PrimeFaces provides a feature to monitor file downloads via client side *monitorDownload(startFunction, endFunction)* method. Example below displays a modal dialog when dowload begins and hides it on complete.

```
<script type="text/javascript">
      function showStatus() {
      statusDialog.show();
      }

      function hideStatus() {
      statusDialog.hide();
      }
</script>
```

```
<h:form>

    <p:dialog modal="true" widgetVar="statusDialog" header="Status" draggable="false"
      closable="false">
        <p:graphicImage value="/design/ajaxloadingbar.gif" />
    </p:dialog>

    <p:commandButton value="Download" ajax="false"
      onclick="PrimeFaces.monitorDownload(showStatus, hideStatus)">
        <p:fileDownload value="#{fileDownloadController.file}"/>
    </p:commandButton>

</h:form>
```

3.36 FileUpload

FileUpload goes beyond the browser input type="file" functionality and features an html5 powered rich solution with graceful degradation for legacy browsers.

Info

Tag	**fileUpload**
Component Class	**org.primefaces.component.fileupload.FileUpload**
Component Type	**org.primefaces.component.FileUpload**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.FileUploadRenderer**
Renderer Class	**org.primefaces.component.fileupload.FileUploadRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	Object	Value of the component than can be either an EL expression of a literal text.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id.
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required.
validator	null	MethodExpr	A method expression that refers to a method validationg the input.

Name	Default	Type	Description
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuchangeevent.
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fails.
widgetVar	null	String	Name of the client side widget.
update	null	String	Component(s) to update after fileupload completes.
process	null	String	Component(s) to process in fileupload request.
fileUploadListener	null	MethodExpr	Method to invoke when a file is uploaded.
multiple	FALSE	Boolean	Allows multi file selection when true.
auto	FALSE	Boolean	Enables auto file uploads.
label	Choose	String	Label of the browse button.
allowTypes	null	String	Regular expression to restrict uploadable files.
sizeLimit	null	Integer	Maximium file size limit in bytes.
fileLimit	null	Integer	Maximum number of files allowed to upload.
showButtons	TRUE	Boolean	Visibility of upload and cancel buttons in button bar.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
mode	advanced	String	Mode of the fileupload, can be *simple* or *advanced*.
uploadLabel	Upload	String	Label of the upload button.
cancelLabel	Cancel	String	Label of the cancel button.
invalidSizeMessage	null	String	Message to display when size limit exceeds.
invalidFileMessage	null	String	Message to display when file is not accepted.
fileLimitMessage	null	String	Message to display when file limit exceeds.
dragDropSupport	TRUE	Boolean	Whether or not to enable dragdrop from filesystem.
onstart	null	String	Client side callback to execute when upload begins.
oncomplete	null	String	Client side callback to execute when upload ends.
disabled	FALSE	Boolean	Disables component when set true.

Getting started with FileUpload

First thing to do is to configure the fileupload filter which parses the multipart request. FileUpload filter should map to Faces Servlet.

```
<filter>
      <filter-name>PrimeFaces FileUpload Filter</filter-name>
      <filter-class>
            org.primefaces.webapp.filter.FileUploadFilter
      </filter-class>
</filter>
<filter-mapping>
      <filter-name>PrimeFaces FileUpload Filter</filter-name>
      <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

Simple File Upload

Simple file upload mode works in legacy mode with a file input whose value should be an UploadedFile instance.

```
<h:form enctype="multipart/form-data">
      <p:fileUpload value="#{fileBean.file}" mode="simple" />

      <p:commandButton value="Submit" ajax="false"/>
</h:form>
```

```
import org.primefaces.model.UploadedFile;

public class FileBean {

      private UploadedFile file;

      //getter-setter
}
```

Advanced File Upload

Default mode of fileupload is advanced that provides a richer UI. In this case, FileUploadListener is the way to access the uploaded files, when a file is uploaded defined fileUploadListener is processed with a FileUploadEvent as the parameter.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}" />
```

```
public class FileBean {

      public void handleFileUpload(FileUploadEvent event) {
            UploadedFile file = event.getFile();
            //application code
      }
}
```

Multiple Uploads

Multiple uploads can be enabled using the multiple attribute. This way multiple files can be selected and uploaded together.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}"  multiple="true" />
```

Auto Upload

Default behavior requires users to trigger the upload process, you can change this way by setting auto to true. Auto uploads are triggered as soon as files are selected from the dialog.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}"  auto="true" />
```

Partial Page Update

After the fileUpload process completes you can use the PrimeFaces PPR to update any component on the page. FileUpload is equipped with the update attribute for this purpose. Following example displays a "File Uploaded" message using the growl component after file upload.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}" update="msg" />

<p:growl id="msg" />
```

```
public class FileBean {

      public void handleFileUpload(FileUploadEvent event) {
            //add facesmessage to display with growl
            //application code
      }
}
```

File Filters

Users can be restricted to only select the file types you've configured, example below demonstrates how to accept images only.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}"
     allowTypes="/(\.|\/)(gif|jpe?g|png)$/" description="Select Images"/>
```

Size Limit

Most of the time you might need to restrict the file upload size, this is as simple as setting the sizeLimit configuration. Following fileUpload limits the size to 1000 bytes for each file.

```
<p:fileUpload fileUploadListener="#{fileBean.handleFileUpload}"  sizeLimit="1000" />
```

Skinning FileUpload

FileUpload resides in a container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Class	Applies
.ui-fileupload	Main container element
.fileupload-buttonbar	Button bar.
.fileinput-button	Browse button.
.ui-fileupload start	Upload button.
.ui-fileupload cancel	Cancel button.
fileupload-content	Content container.

As skinning style classes are global, see the main Skinning section for more information.

Browser Compatibility

Rich upload functionality like dragdrop from filesystem, multi uploads, progress tracking requires browsers that implement HTML5 features so advanced mode might behave differently across browsers and gracefully degrade for legacy browsers like IE. It is also suggested to offer simple upload mode to the users of your application as a fallback.

Filter Configuration

FileUpload filter's default settings can be configured with init parameters. Two configuration options exist, threshold size and temporary file upload location.

Parameter Name	Description
thresholdSize	Maximum file size in bytes to keep uploaded files in memory. If a file exceeds this limit, it'll be temporarily written to disk.
uploadDirectory	Disk repository path to keep temporary files that exceeds the threshold size. By default it is System.getProperty("java.io.tmpdir")

An example configuration below defined thresholdSize to be 50kb and uploads to user's temporary folder.

```
<filter>
      <filter-name>PrimeFaces FileUpload Filter</filter-name>
      <filter-class>
            org.primefaces.webapp.filter.FileUploadFilter
      </filter-class>
      <init-param>
            <param-name>thresholdSize</param-name>
            <param-value>51200</param-value>
      </init-param>
      <init-param>
            <param-name>uploadDirectory</param-name>
            <param-value>/Users/primefaces/temp</param-value>
      </init-param>
</filter>
```

Note that uploadDirectory is used internally, you should implement the logic to save the file contents yourself in your backing bean.

Tips

- Multiple advanced uploaders in same form is not supported at the moment.

3.37 Focus

Focus is a utility component that makes it easy to manage the element focus on a JSF page.

Info

Tag	**focus**
Component Class	**org.primefaces.component.focus.Focus**
Component Type	**org.primefaces.component.Focus.FocusTag**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.FocusRenderer**
Renderer Class	**org.primefaces.component.focus.FocusRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
for	null	String	Specifies the exact component to set focus
context	null	String	The root component to start first input search.
minSeverity	error	String	Minimum severity level to be used when finding the first invalid component

Getting started with Focus

By default focus will find the *first enabled and visible input component* on page and apply focus. Input component can be any element such as input, textarea and select.

```
<p:focus />

<p:inputText ... />
<h:inputText ... />
<h:selectOneMenu ... />
```

Following is a simple example;

```
<h:form>
  <p:panel id="panel" header="Register">

      <p:focus />

      <p:messages />

      <h:panelGrid columns="3">
            <h:outputLabel for="firstname" value="Firstname: *" />
            <h:inputText id="firstname" value="#{pprBean.firstname}"
                         required="true" label="Firstname" />
            <p:message for="firstname" />

            <h:outputLabel for="surname" value="Surname: *" />
            <h:inputText id="surname" value="#{pprBean.surname}"
                         required="true" label="Surname"/>
            <p:message for="surname" />
      </h:panelGrid>

      <p:commandButton value="Submit" update="panel"
                  actionListener="#{pprBean.savePerson}" />
  </p:panel>
</h:form>
```

When this page initially opens, input text with id "firstname" will receive focus as it is the first input component.

Validation Aware

Another useful feature of focus is that when validations fail, *first invalid component* will receive a focus. So in previous example if firstname field is valid but surname field has no input, a validation error will be raised for surname, in this case focus will be set on surname field implicitly. Note that for this feature to work on ajax requests, you need to update p:focus component as well.

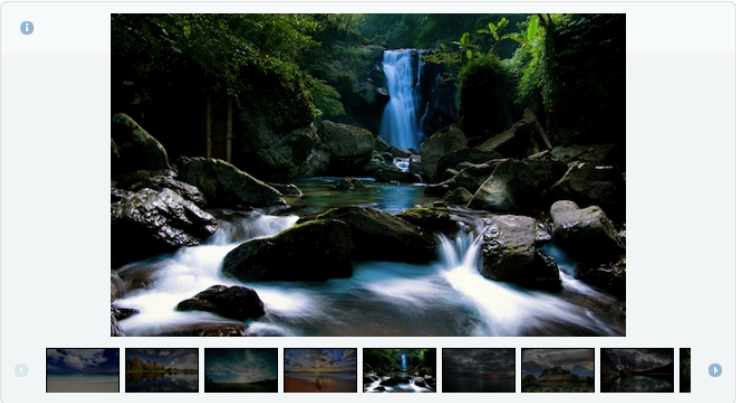
Explicit Focus

Additionally, using for attribute focus can be set explicitly on an input component which is useful when using a dialog.

```
<p:focus for="text"/>

<h:inputText id="text" value="{bean.value}" />
```

3.38 Galleria

Galleria is used to display a set of images.



Info

Tag	**galleria**
Component Class	**org.primefaces.component.galleria.Galleria**
Component Type	**org.primefaces.component.Galleria**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.GalleriaRenderer**
Renderer Class	**org.primefaces.component.galleria.GalleriaRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
value	null	Collection	Collection of data to display.
var	null	String	Name of variable to access an item in collection.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.

Name	Default	Type	Description
effect	fade	String	Name of animation to use.
effectSpeed	700	Integer	Duration of animation in milliseconds.
panelWidth	600	Integer	Width of the viewport.
panelHeight	400	Integer	Height of the viewport.
frameWidth	60	Integer	Width of the frames.
frameHeight	40	Integer	Height of the frames.
filmstripStyle	null	String	Style of the filmstrip.
filmstripPosition	null	String	Position of the filmstrip.
showFilmstrip	TRUE	Boolean	Defines visibility of filmstrip.
showCaptions	FALSE	Boolean	Defines visibility of captions.
showOverlays	FALSE	Boolean	Defines visibility of overlays.
transitionInterval	4000	Integer	Defines interval of slideshow.

Getting Started with Galleria

Images to displayed are defined as children of galleria;

```
<p:galleria effect="slide" effectDuration="1000">
     <p:graphicImage value="/images/image1.jpg" title="image1" alt="image1 desc" />
     <p:graphicImage value="/images/image2.jpg" title="image1" alt=" image2 desc" />
     <p:graphicImage value="/images/image3.jpg" title="image1" alt=" image3 desc" />
     <p:graphicImage value="/images/image4.jpg" title="image1" alt=" image4 desc" />
</p:galleria>
```

Galleria displays the details of an image using an overlay which is displayed by clicking the information icon. Title of this popup is retrieved from the image *title* attribute and description from *alt* attribute so it is suggested to provide these attributes as well.

Dynamic Collection

Most of the time, you would need to display a dynamic set of images rather than defining each image declaratively. For this you can use built-in data iteration feature.

```
<p:galleria value="#{galleriaBean.images}" var="image" >
     <p:graphicImage value="#{image.path}"
                     title="#{image.title}" alt="#{image.description}" />
 </p:galleria>
```

Effects

There are four different options for the animation to display when switching images;

- fade (default)
- flash
- pulse
- slide

By default animation takes 700 milliseconds, use *effectDuration* option to tune this.

```
<p:galleria effect="slide" effectDuration="1000">
     <p:graphicImage value="/images/image1.jpg" title="image1" alt="image1 desc" />
     <p:graphicImage value="/images/image2.jpg" title="image1" alt=" image2 desc" />
     <p:graphicImage value="/images/image3.jpg" title="image1" alt=" image3 desc" />
     <p:graphicImage value="/images/image4.jpg" title="image1" alt=" image4 desc" />
</p:galleria>
```

Skinning

Galleria resides in a main container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-galleria-container	Container element for galleria.
.ui-galleria-stage	Container displaying actual images.
.ui-galleria-thumbnails-container	Container displaying thumbnail images.
.ui-galleria-thumbnails-list	Thumbnail images list
.ui-galleria-thumbnails .ui-galleria-image	Each thumbnail in list
.ui-galleria-counter	Text showing image index/total
.ui-galleria-info	Info overlay container
.ui-galleria-text	Text in info overlay.
.ui-galleria-title	Info title
.ui-galleria-description	Info description
.ui-galleria-image-thumb-nav-left	Left thumbnail navigator
.ui-galleria-image-thumb-nav-right	Right thumbnail navigator

As skinning style classes are global, see the main Skinning section for more information.

3.39 GMap

GMap is a map component integrated with Google Maps API V3.

Info

Tag	**gmap**
Component Class	**org.primefaces.component.gmap.GMap**
Component Type	**org.primefaces.component.Gmap**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.GmapRenderer**
Renderer Class	**org.primefaces.component.gmap.GmapRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget.

Name	Default	Type	Description
model	null	MapModel	An org.primefaces.model.MapModel instance.
style	null	String	Inline style of the map container.
styleClass	null	String	Style class of the map container.
type	null	String	Type of the map.
center	null	String	Center point of the map.
zoom	8	Integer	Defines the initial zoom level.
streetView	FALSE	Boolean	Controls street view support.
disableDefaultUI	FALSE	Boolean	Disables default UI controls
navigationControl	TRUE	Boolean	Defines visibility of navigation control.
mapTypeControl	TRUE	Boolean	Defines visibility of map type control.
draggable	TRUE	Boolean	Defines draggability of map.
disabledDoubleClickZoom	FALSE	Boolean	Disables zooming on mouse double click.
onPointClick	null	String	Javascript callback to execute when a point on map is clicked.
fitBounds	TRUE	Boolean	Defines if center and zoom should be calculated automatically to contain all markers on the map.

Getting started with GMap

First thing to do is placing V3 of the Google Maps API that the GMap is based on. Ideal location is the head section of your page.

```
<script src="http://maps.google.com/maps/api/js?sensor=true|false"
     type="text/javascript"></script>
```

As Google Maps api states, mandatory sensor parameter is used to specify if your application requires a sensor like GPS locator. Four options are required to place a gmap on a page, these are center, zoom, type and style.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
                                style="width:600px;height:400px" />
```

center: Center of the map in lat, lng format
zoom: Zoom level of the map

type: Type of map, valid values are, "hybrid", "satellite", "hybrid" and "terrain".
style: Dimensions of the map.

MapModel

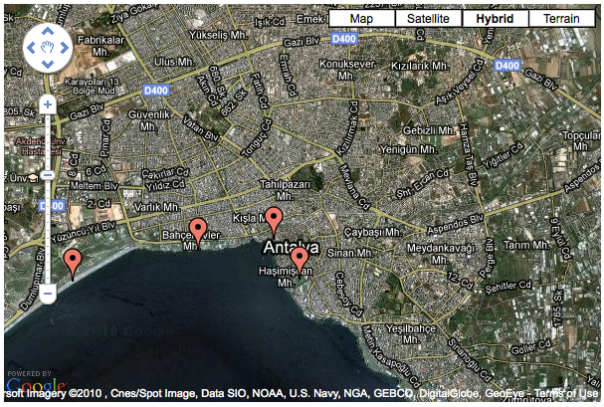
GMap is backed by an *org.primefaces.model.map.MapModel* instance, PrimeFaces provides *org.primefaces.model.map.DefaultMapModel* as the default implementation. API Docs of all GMap related model classes are available at the end of GMap section and also at javadocs of PrimeFaces.

Markers

A marker is represented by *org.primefaces.model.map.Marker*.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" model="#{mapBean.model}"/>
```

```
public class MapBean {

   private MapModel model = new DefaultMapModel();

   public MapBean() {
       model.addOverlay(new Marker(new LatLng(36.879466, 30.667648), "M1"));
       //more overlays
   }

   public MapModel getModel() { return this.model; }
}
```



Polylines

A polyline is represented by *org.primefaces.model.map.Polyline*.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" model="#{mapBean.model}"/>
```

```
public class MapBean {

  private MapModel model;

  public MapBean() {
      model = new DefaultMapModel();

      Polyline polyline = new Polyline();
      polyline.getPaths().add(new LatLng(36.879466, 30.667648));
      polyline.getPaths().add(new LatLng(36.883707, 30.689216));
      polyline.getPaths().add(new LatLng(36.879703, 30.706707));
      polyline.getPaths().add(new LatLng(36.885233, 37.702323));

      model.addOverlay(polyline);
  }

  public MapModel getModel() { return this.model; }
}
```

Polygons

A polygon is represented by *org.primefaces.model.map.Polygon*.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" model="#{mapBean.model}"/>
```

```
public class MapBean {

  private MapModel model;

  public MapBean() {
      model = new DefaultMapModel();

      Polygon polygon = new Polygon();
      polyline.getPaths().add(new LatLng(36.879466, 30.667648));
      polyline.getPaths().add(new LatLng(36.883707, 30.689216));
      polyline.getPaths().add(new LatLng(36.879703, 30.706707));

      model.addOverlay(polygon);
  }

  public MapModel getModel() { return this.model; }
}
```

Circles

A circle is represented by *org.primefaces.model.map.Circle*.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" model="#{mapBean.model}"/>
```

```
public class MapBean {

  private MapModel model;

  public MapBean() {
      model = new DefaultMapModel();

      Circle circle = new Circle(new LatLng(36.879466, 30.667648), 500);

      model.addOverlay(circle);
  }

  public MapModel getModel() { return this.model; }
}
```

Rectangles

A circle is represented by *org.primefaces.model.map.Rectangle*.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" model="#{mapBean.model}"/>
```

```
public class MapBean {

  private MapModel model;

  public MapBean() {
      model = new DefaultMapModel();
      LatLng coord1 = new LatLng(36.879466, 30.667648);
      LatLng coord2 = new LatLng(36.883707, 30.689216);

      Rectangle rectangle = new Rectangle(coord1, coord2);

      model.addOverlay(circle);
  }

  public MapModel getModel() { return this.model; }
}
```

Ajax Behavior Events

GMap provides many custom ajax behavior events for you to hook-in to various features.

Event	Listener Parameter	Fired
overlaySelect	org.primefaces.event.map.OverlaySelectEvent	When an overlay is selected.
stateChange	org.primefaces.event.map.StateChangeEvent	When map state changes.
pointSelect	org.primefaces.event.map.PointSelectEvent	When an empty point is selected.
markerDrag	org.primefaces.event.map.MarkerDragEvent	When a marker is dragged.

Following example displays a FacesMessage about the selected marker with growl component.

```
<h:form>
  <p:growl id="growl" />

  <p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
          style="width:600px;height:400px" model="#{mapBean.model}">
     <p:ajax event="overlaySelect" listener="#{mapBean.onMarkerSelect}"
          update="growl" />
  </p:gmap>
</h:form>
```

```
public class MapBean {

  private MapModel model;

  public MapBean() {
     model = new DefaultMapModel();
     //add markers
  }

  public MapModel getModel() {
     return model
  }

  public void onMarkerSelect(OverlaySelectEvent event) {
     Marker selectedMarker = (Marker) event.getOverlay();
     //add facesmessage
  }
}
```

InfoWindow

A common use case is displaying an info window when a marker is selected. *gmapInfoWindow* is used to implement this special use case. Following example, displays an info window that contains an image of the selected marker data.

```
<h:form>

  <p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
          style="width:600px;height:400px" model="#{mapBean.model}">

     <p:ajax event="overlaySelect" listener="#{mapBean.onMarkerSelect}" />

     <p:gmapInfoWindow>
          <p:graphicImage value="/images/#{mapBean.marker.data.image}" />
          <h:outputText value="#{mapBean.marker.data.title}" />
     </p:gmapInfoWindow>
  </p:gmap>

</h:form>
```

```
public class MapBean {

  private MapModel model;

  private Marker marker;

  public MapBean() {
      model = new DefaultMapModel();
      //add markers
  }

  public MapModel getModel() { return model; }

  public Marker getMarker() { return marker; }

  public void onMarkerSelect(OverlaySelectEvent event) {
      this.marker = (Marker) event.getOverlay();
  }
}
```

Street View

StreeView is enabled simply by setting *streetView* option to true.

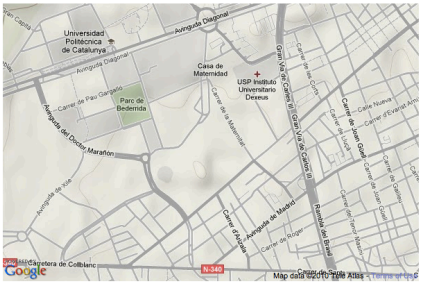
```
<p:gmap center="41.381542, 2.122893" zoom="15" type="hybrid"
            style="width:600px;height:400px" streetView="true" />
```

Avinguda Litoral / Ronda del Litoral, Barcelona, CT
Address is approximate
Avinguda Litoral
Ronda del Litoral
Avinguda Litoral
© 2010 Google - Terms of Use

Map Controls

Controls on map can be customized via attributes like *navigationControl* and *mapTypeControl*. Alternatively setting *disableDefaultUI* to true will remove all controls at once.

```
<p:gmap center="41.381542, 2.122893" zoom="15" type="terrain"
            style="width:600px;height:400px"
```



Native Google Maps API

In case you need to access native google maps api with javascript, use provided *getMap()* method.

```
 var gmap = yourWidgetVar.getMap();
//gmap is a google.maps.Map instance
```

Full map api is provided at;

http://code.google.com/apis/maps/documentation/javascript/reference.html

GMap API

org.primefaces.model.map.MapModel (*org.primefaces.model.map.DefaultMapModel* is the default implementation)

Method	Description
addOverlay(Overlay overlay)	Adds an overlay to map
List<Marker> getMarkers()	Returns the list of markers
List<Polyline> getPolylines()	Returns the list of polylines
List<Polygon> getPolygons()	Returns the list of polygons
List<Circle> getCircles()	Returns the list of circles
List<Rectangle> getRectangles()	Returns the list of rectangles.
Overlay findOverlay(String id)	Finds an overlay by it's unique id

org.primefaces.model.map.Overlay

Property	Default	Type	Description
id	null	String	Id of the overlay, generated and used internally
data	null	Object	Data represented in marker
zindex	null	Integer	Z-Index of the overlay

org.primefaces.model.map.Marker extends *org.primefaces.model.map.Overlay*

Property	Default	Type	Description
title	null	String	Text to display on rollover
latlng	null	LatLng	Location of the marker
icon	null	String	Icon of the foreground
shadow	null	String	Shadow image of the marker
cursor	pointer	String	Cursor to display on rollover
draggable	FALSE	Boolean	Defines if marker can be dragged
clickable	TRUE	Boolean	Defines if marker can be dragged
flat	FALSE	Boolean	If enabled, shadow image is not displayed
visible	TRUE	Boolean	Defines visibility of the marker

org.primefaces.model.map.Polyline extends *org.primefaces.model.map.Overlay*

Property	Default	Type	Description
paths	null	List	List of coordinates
strokeColor	null	String	Color of a line
strokeOpacity	1	Double	Opacity of a line
strokeWeight	1	Integer	Width of a line

org.primefaces.model.map.Polygon extends *org.primefaces.model.map.Overlay*

Property	Default	Type	Description
paths	null	List	List of coordinates
strokeColor	null	String	Color of a line

Property	Default	Type	Description
strokeOpacity	1	Double	Opacity of a line
strokeWeight	1	Integer	Weight of a line
fillColor	null	String	Background color of the polygon
fillOpacity	1	Double	Opacity of the polygon

org.primefaces.model.map.Circle extends *org.primefaces.model.map.Overlay*

Property	Default	Type	Description
center	null	LatLng	Center of the circle
radius	null	Double	Radius of the circle.
strokeColor	null	String	Stroke color of the circle.
strokeOpacity	1	Double	Stroke opacity of circle.
strokeWeight	1	Integer	Stroke weight of the circle.
fillColor	null	String	Background color of the circle.
fillOpacity	1	Double	Opacity of the circle.

org.primefaces.model.map.Rectangle extends *org.primefaces.model.map.Overlay*

Property	Default	Type	Description
bounds	null	LatLngBounds	Boundaries of the rectangle.
strokeColor	null	String	Stroke color of the rectangle.
strokeOpacity	1	Double	Stroke opacity of rectangle.
strokeWeight	1	Integer	Stroke weight of the rectangle.
fillColor	null	String	Background color of the rectangle.
fillOpacity	1	Double	Opacity of the rectangle.

org.primefaces.model.map.LatLng

Property	Default	Type	Description
lat	null	double	Latitude of the coordinate
lng	null	double	Longitude of the coordinate

org.primefaces.model.map.LatLngBounds

Property	Default	Type	Description
center	null	LatLng	Center coordinate of the boundary
northEast	null	LatLng	NorthEast coordinate of the boundary
southWest	null	LatLng	SouthWest coordinate of the boundary

GMap Event API

All classes in event api extends from *javax.faces.event.FacesEvent*.

org.primefaces.event.map.MarkerDragEvent

Property	Default	Type	Description
marker	null	Marker	Dragged marker instance

org.primefaces.event.map.OverlaySelectEvent

Property	Default	Type	Description
overlay	null	Overlay	Selected overlay instance

org.primefaces.event.map.PointSelectEvent

Property	Default	Type	Description
latLng	null	LatLng	Coordinates of the selected point

org.primefaces.event.map.StateChangeEvent

Property	Default	Type	Description
bounds	null	LatLngBounds	Boundaries of the map
zoomLevel	0	Integer	Zoom level of the map

3.40 GMapInfoWindow

GMapInfoWindow is used with GMap component to open a window on map when an overlay is selected.

Info

Tag	**gmapInfoWindow**
Tag Class	**org.primefaces.component.gmap.GMapInfoWindowTag**
Component Class	**org.primefaces.component.gmap.GMapInfoWindow**
Component Type	**org.primefaces.component.GMapInfoWindow**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
maxWidth	null	Integer	Maximum width of the info window

Getting started with GMapInfoWindow

See GMap section for more information about how gmapInfoWindow is used.

3.41 GraphicImage

PrimeFaces GraphicImage extends standard JSF graphic image component with the ability of displaying binary data like an inputstream. Main use cases of GraphicImage is to make displaying images stored in database or on-the-fly images easier. Legacy way to do this is to come up with a Servlet that does the streaming, GraphicImage does all the hard work without the need of a Servlet.

Info

Tag	**graphicImage**
Component Class	**org.primefaces.component.graphicimage.GraphicImage**
Component Type	**org.primefaces.component.GraphicImage**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.GraphicImageRenderer**
Renderer Class	**org.primefaces.component.graphicimage.GraphicImageRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Binary data to stream or context relative path.
alt	null	String	Alternate text for the image
url	null	String	Alias to value attribute
width	null	String	Width of the image
height	null	String	Height of the image
title	null	String	Title of the image
dir	null	String	Direction of the text displayed
lang	null	String	Language code
ismap	FALSE	Boolean	Specifies to use a server-side image map
usemap	null	String	Name of the client side map
style	null	String	Style of the image

Name	Default	Type	Description
styleClass	null	String	Style class of the image
onclick	null	String	onclick dom event handler
ondblclick	null	String	ondblclick dom event handler
onkeydown	null	String	onkeydown dom event handler
onkeypress	null	String	onkeypress dom event handler
onkeyup	null	String	onkeyup dom event handler
onmousedown	null	String	onmousedown dom event handler
onmousemove	null	String	onmousemove dom event handler
onmouseout	null	String	onmouseout dom event handler
onmouseover	null	String	onmouseover dom event handler
onmouseup	null	String	onmouseup dom event handler
cache	TRUE	String	Enables/Disables browser from caching the image

Getting started with GraphicImage

GraphicImage requires an *org.primefaces.model.StreamedContent* content as it's value. StreamedContent is an interface and PrimeFaces provides a built-in implementation called *DefaultStreamedContent*. Following examples loads an image from the classpath.

```
<p:graphicImage value="#{imageBean.image}" />
```

```
public class ImageBean {

      private StreamedContent image;

      public DynamicImageController() {
         InputStream stream = this.getClass().getResourceAsStream("barcalogo.jpg");
         image = new DefaultStreamedContent(stream, "image/jpeg");
      }

      public StreamedContent getImage() {
            return this.image;
      }
}
```

DefaultStreamedContent gets an inputstream as the first parameter and mime type as the second.

In a real life application, you can create the inputstream after reading the image from the database. For example *java.sql.ResultsSet* API has the getBinaryStream() method to read blob files stored in database.

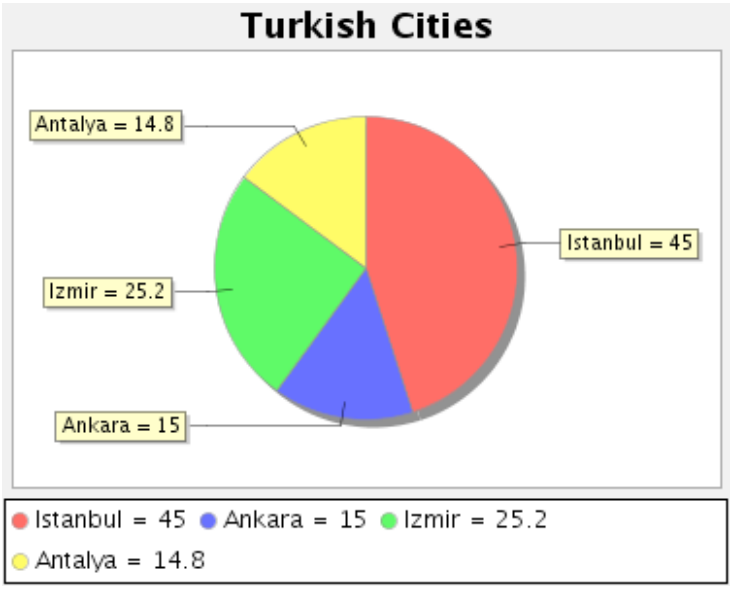
Displaying Charts with JFreeChart

StreamedContent is a powerful API that can display images created on-the-fly as well. Here's an example that generates a chart with JFreeChart and displays it with p:graphicImage.

```
<p:graphicImage value="#{chartBean.chart}" />
```

```
public class ChartBean {

      private StreamedContent chart;

      public BackingBean() {
            try {
                  JFreeChart jfreechart = ChartFactory.createPieChart(
                        "Turkish Cities", createDataset(), true, true, false);
                  File chartFile = new File("dynamichart");
                  ChartUtilities.saveChartAsPNG(chartFile, jfreechart, 375, 300);
                  chart = new DefaultStreamedContent(
                        new FileInputStream(chartFile), "image/png");
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }

      private PieDataset createDataset() {
            DefaultPieDataset dataset = new DefaultPieDataset();
            dataset.setValue("Istanbul", new Double(45.0));
            dataset.setValue("Ankara", new Double(15.0));
            dataset.setValue("Izmir", new Double(25.2));
            dataset.setValue("Antalya", new Double(14.8));

            return dataset;
      }

      public StreamedContent getChart() {
            return this.chart;
      }
}
```

Basically p:graphicImage makes any JSF chart component using JFreechart obsolete and lets you to avoid wrappers(e.g. JSF ChartCreator project which we've created in the past) to take full advantage of JFreechart API.



Displaying a Barcode

Similar to the chart example, a barcode can be generated as well. This sample uses barbecue project for the barcode API.

```
<p:graphicImage value="#{backingBean.barcode}" />
```

```
public class BarcodeBean {

      private StreamedContent barcode;

      public BackingBean() {
            try {
                  File barcodeFile = new File("dynamicbarcode");
                  BarcodeImageHandler.saveJPEG(
                        BarcodeFactory.createCode128("PRIMEFACES"), barcodeFile);
                  barcode = new DefaultStreamedContent(
                        new FileInputStream(barcodeFile), "image/jpeg");
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }

      public BarcodeBean getBarcode() {
            return this.barcode;
      }
}
```

Displaying Regular Images

As GraphicImage extends standard graphicImage component, it can also display regular non dynamic images.

```
<p:graphicImage value="barcalogo.jpg" />
```

How It Works

Dynamic image display works as follows;

- Dynamic image puts its value expression string to the http session with a unique key.
- Unique session key is appended to the image url that points to JSF resource handler.
- Custom PrimeFaces ResourceHandler get the key from the url, retrieves the expression string like #{bean.streamedContentValue}, evaluates it to get the instance of StreamedContent from bean and streams contents to client.

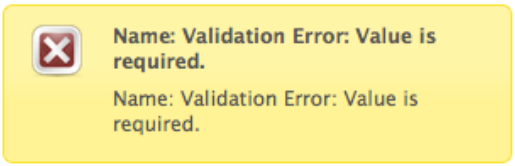
As a result there will be 2 requests to display an image, first browser will make a request to load the page and then another one to the dynamic image url that points to JSF resource handler. Please note that you cannot use viewscope beans as viewscoped bean is not available in resource loading request.

Passing Parameters and Data Iteration

You can pass request parameters to the graphicImage via f:param tags, as a result the actual request rendering the image can have access to these values. This is extremely handy to display dynamic images if your image is in a data iteration component like datatable or ui:repeat.

3.42 Growl

Growl is based on the Mac's growl notification widget and used to display FacesMessages in an overlay.



Info

Tag	**growl**
Component Class	**org.primefaces.component.growl.Growl**
Component Type	**org.primefaces.component.Growl**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.GrowlRenderer**
Renderer Class	**org.primefaces.component.growl.GrowlRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
sticky	FALSE	Boolean	Specifies if the message should stay instead of hidden automatically.
showSummary	TRUE	Boolean	Specifies if the summary of message should be displayed.
showDetail	FALSE	Boolean	Specifies if the detail of message should be displayed.
globalOnly	FALSE	Boolean	When true, only facesmessages without clientids are displayed.
life	6000	Integer	Duration in milliseconds to display non-sticky messages.
autoUpdate	FALSE	Boolean	Specifies auto update mode.

Name	Default	Type	Description
redisplay	TRUE	Boolean	Defines if already rendered messaged should be displayed.
for	null	String	Name of associated key, takes precedence when used with globalOnly.
escape	TRUE	Boolean	Defines whether html would be escaped or not.
severity	null	String	Comma separated list of severities to display only.

Getting Started with Growl

Growl usage is similar to standard h:messages component. Simply place growl anywhere on your page, since messages are displayed as an overlay, the location of growl in JSF page does not matter.

```
<p:growl />
```

Lifetime of messages

By default each message will be displayed for 6000 ms and then hidden. A message can be made sticky meaning it'll never be hidden automatically.

```
<p:growl sticky="true" />
```

If growl is not working in sticky mode, it's also possible to tune the duration of displaying messages. Following growl will display the messages for 5 seconds and then fade-out.

```
<p:growl life="5000" />
```

Growl with Ajax Updates

If you need to display messages with growl after an ajax request you just need to update it. Note that if you enable autoUpdate, growl will be updated automatically with each ajax request anyway.

```
<p:growl id="messages"/>

<p:commandButton value="Submit" update="messages" />
```

Positioning

Growl is positioned at top right corner by default, position can be controlled with a CSS selector called *ui-growl*. With the below setting growl will be located at top left corner.

```
.ui-growl {
      left:20px;
}
```

Targetable Messages

There may be times where you need to target one or more messages to a specific message component, for example suppose you have growl and messages on same page and you need to display some messages on growl and some on messages. Use for attribute to associate messages with specific components.

```
<p:messages for="somekey" />
<p:growl for="anotherkey" />
```

```
FacesContext context = FacesContext.getCurrentInstance();

context.addMessage("somekey", facesMessage1);
context.addMessage("somekey", facesMessage2);

context.addMessage("anotherkey", facesMessage3);
```

In sample above, messages will display first and second message and growl will only display the 3rd message.

Severity Levels

Using severity attribute, you can define which severities can be displayed by the component. For instance, you can configure growl to only display infos and warnings.

```
<p:growl severity="info,warn" />
```

Escaping

Growl escapes html content in messages, in case you need to display html via growl set escape option to true.

```
<p:growl escape="true" />
```

Skinning

Following is the list of structural style classes;

Style Class	Applies
.ui-growl	Main container element of growl
.ui-growl-item-container	Container of messages
.ui-growl-item	Container of a message
.ui-growl-message	Text message container
.ui-growl-title	Summary of the message
.ui-growl-message p	Detail of the message
.ui-growl-image	Severity icon
.ui-growl-image-info	Info severity icon
.ui-growl-image-warn	Warning severity icon
.ui-growl-image-error	Error severity icon
.ui-growl-image-fatal	Fatal severity icon

As skinning style classes are global, see the main Skinning section for more information.

3.43 HotKey

HotKey is a generic key binding component that can bind any formation of keys to javascript event handlers or ajax calls.

Info

Tag	**hotkey**
Component Class	**org.primefaces.component.hotkey.HotKey**
Component Type	**org.primefaces.component.HotKey**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.HotKeyRenderer**
Renderer Class	**org.primefaces.component.hotkey.HotKeyRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
bind	null	String	The Key binding.
handler	null	String	Javascript event handler to be executed when the key binding is pressed.
action	null	MethodExpr	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	MethodExpr	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component id(s) to process partially instead of whole view.
update	null	String	Client side id of the component(s) to be updated after async partial submit request.

Name	Default	Type	Description
onstart	null	String	Javascript handler to execute before ajax request is begins.
oncomplete	null	String	Javascript handler to execute when ajax request is completed.
onsuccess	null	String	Javascript handler to execute when ajax request succeeds.
onerror	null	String	Javascript handler to execute when ajax request fails.
global	TRUE	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.

Getting Started with HotKey

HotKey is used in two ways, either on client side with the event handler or with ajax support. Simplest example would be;

```
<p:hotkey bind="a" handler="alert('Pressed a');" />
```

When this hotkey is on page, pressing the a key will alert the 'Pressed key a' text.

Key combinations

Most of the time you'd need key combinations rather than a single key.

```
<p:hotkey bind="ctrl+s" handler="alert('Pressed ctrl+s');" />
<p:hotkey bind="ctrl+shift+s" handler="alert('Pressed ctrl+shift+s')" />
```

Integration

Here's an example demonstrating how to integrate hotkeys with a client side api. Using left and right keys will switch the images displayed via the p:imageSwitch component.

```
<p:hotkey bind="left" handler="switcher.previous();" />
<p:hotkey bind="right" handler="switcher.next();" />

<p:imageSwitch widgetVar="switcher">
      //content
</p:imageSwitch>
```

Ajax Support

Ajax is a built-in feature of hotKeys meaning you can do ajax calls with key combinations. Following form can be submitted with the *ctrl+shift+s* combination.

```
<h:form>

  <p:hotkey bind="ctrl+shift+s" update="display" />

  <h:panelGrid columns="2">
      <h:outputLabel for="name" value="Name:" />
      <h:inputText id="name" value="#{bean.name}" />
  </h:panelGrid>

  <h:outputText id="dsplay" value="Hello: #{bean.firstname}" />

</h:form>
```

Note that hotkey will not be triggered if there is a focused input on page.

3.44 IdleMonitor

IdleMonitor watches user actions on a page and notify callbacks in case they go idle or active again.

Info

Tag	**idleMonitor**
Component Class	**org.primefaces.component.idlemonitor.IdleMonitor**
Component Type	**org.primefaces.component.IdleMonitor**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.IdleMonitorRenderer**
Renderer Class	**org.primefaces.component.idlemonitor.IdleMonitor**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
timeout	300000	Integer	Time to wait in milliseconds until deciding if the user is idle. Default is 5 minutes.
onidle	null	String	Client side callback to execute when user goes idle.
onactive	null	String	Client side callback to execute when user goes idle.
widgetVar	null	String	Name of the client side widget.

Getting Started with IdleMonitor

To begin with, you can hook-in to client side events that are called when a user goes idle or becomes active again. Example below toggles visibility of a dialog to respond these events.

```
<p:idleMonitor onidle="idleDialog.show();" onactive="idleDialog.hide();"/>

<p:dialog header="What's happening?" widgetVar="idleDialog" modal="true">
     <h:outputText value="Dude, are you there?" />
</p:dialog>
```

Controlling Timeout

By default, idleMonitor waits for 5 minutes (300000 ms) until triggering the onidle event. You can customize this duration with the timeout attribute.

Ajax Behavior Events

IdleMonitor provides two ajax behavior events which are *idle* and *active* that are fired according to user status changes. Example below displays messages for each event;

```
<p:idleMonitor timeout="5000" update="messages">
   <p:ajax event="idle" listener="#{bean.idleListener}" update="msg" />
   <p:ajax event="active" listener="#{bean.activeListener}" update="msg" />
</p:idleMonitor>

<p:growl id=”msg” />
```

```
public class Bean {

      public void idleListener() {
            //Add facesmessage
      }

      public void idle() {
            //Add facesmessage
      }
}
```

3.45 ImageCompare

ImageCompare provides a rich user interface to compare two images.

Info

Tag	**imageCompare**
Component Class	**org.primefaces.component.imagecompare.ImageCompare**
Component Type	**org.primefaces.component.ImageCompare**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ImageCompareRenderer**
Renderer Class	**org.primefaces.component.imagecompare.ImageCompareRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Name	Default	Type	Description
widgetVar	null	String	Name of the client side widget.
leftImage	null	String	Source of the image placed on the left side
rightImage	null	String	Source of the image placed on the right side
width	null	String	Width of the images
height	null	String	Height of the images
style	null	String	Inline style of the container element
styleClass	null	String	Style class of the container element

Getting started with ImageCompare

ImageCompare is created with two images with same height and width. It is required to set width and height of the images as well.

```
<p:imageCompare leftImage="xbox.png" rightImage="ps3.png"
                  width="438" height="246"/>
```

Skinning

Both images are placed inside a div container element, *style* and *styleClass* attributes apply to this element.

3.46 ImageCropper

ImageCropper allows cropping a certain region of an image. A new image is created containing the cropped area and assigned to a CroppedImage instanced on the server side.

Info

Tag	**imageCropper**
Component Class	**org.primefaces.component. imagecropper.ImageCropper**
Component Type	**org.primefaces.component.ImageCropper**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ImageCropperRenderer**
Renderer Class	**org.primefaces.component.imagecropper.ImageCropperRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	ValueChange Listener	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
image	null	String	Context relative path to the image.
alt	null	String	Alternate text of the image.
aspectRatio	null	Double	Aspect ratio of the cropper area.
minSize	null	String	Minimum size of the cropper area.
maxSize	null	String	Maximum size of the cropper area.
backgroundColor	null	String	Background color of the container.
backgroundOpacity	0,6	Double	Background opacity of the container
initialCoords	null	String	Initial coordinates of the cropper area.

Getting started with the ImageCropper

ImageCropper is an input component and image to be cropped is provided via the *image* attribute. The cropped area of the original image is used to create a new image, this new image can be accessed on the backing bean by setting the value attribute of the image cropper. Assuming the image is at %WEBAPP_ROOT%/campnou.jpg

```
<p:imageCropper value="#{cropper.croppedImage}" image="/campnou.jpg" />
```

```
public class Cropper {
      private CroppedImage croppedImage;

      //getter and setter
}
```

org.primefaces.model.CroppedImage belongs a PrimeFaces API and contains handy information about the crop process. Following table describes CroppedImage properties.

Property	Type	Description
originalFileName	String	Name of the original file that's cropped
bytes	byte[]	Contents of the cropped area as a byte array
left	int	Left coordinate
right	int	Right coordinate
width	int	Width of the cropped image
height	int	Height of the cropped image

External Images

ImageCropper has the ability to crop external images as well.

```
<p:imageCropper value="#{cropper.croppedImage}"
            image="http://primefaces.prime.com.tr/en/images/schema.png">
</p:imageCropper>
```

Context Relative Path

For local images, ImageCropper always requires the image path to be context relative. So to accomplish this simply just add slash ("/path/to/image.png") and imagecropper will recognize it at %WEBAPP_ROOT%/path/to/image.png. Action url relative local images are not supported.

Initial Coordinates

By default, user action is necessary to initiate the cropper area on an image, you can specify an initial area to display on page load using *initialCoords* option in *x,y,w,h* format.

```
<p:imageCropper value="#{cropper.croppedImage}" image="/campnou.jpg"
            initialCoords="225,75,300,125"/>
```

Boundaries

minSize and maxSize attributes are control to limit the size of the area to crop.

```
<p:imageCropper value="#{cropper.croppedImage}" image="/campnou.jpg"
            minSize="50,100" maxSize="150,200"/>
```

Saving Images

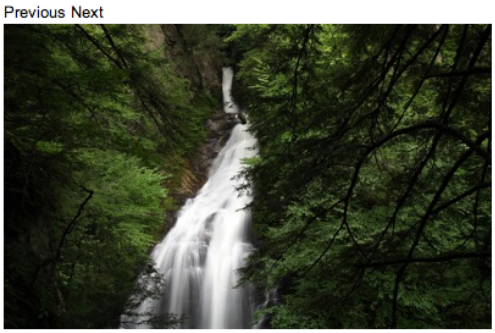
Below is an example to save the cropped image to file system.

```
<p:imageCropper value="#{cropper.croppedImage}" image="/campnou.jpg" />

<p:commandButton value="Crop" actionListener="#{myBean.crop}" />
```

```
public class Cropper {

      private CroppedImage croppedImage;

      //getter and setter

      public String crop() {
            ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
            String newFileName = servletContext.getRealPath("") + File.separator +
"ui" + File.separator + "barca" + File.separator+ croppedImage.getOriginalFileName()
+ "cropped.jpg";

            FileImageOutputStream imageOutput;
            try {
                  imageOutput = new FileImageOutputStream(new File(newFileName));
                  imageOutput.write(croppedImage.getBytes(), 0,
croppedImage.getBytes().length);
                  imageOutput.close();
            } catch (Exception e) {
                  e.printStackTrace();
            }
            return null;
      }
}
```

3.47 ImageSwitch

Imageswitch component is a simple image gallery component.



Info

Tag	**imageSwitch**
Component Class	**org.primefaces.component.imageswitch.ImageSwitch**
Component Type	**org.primefaces.component.ImageSwitch**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ImageSwitchRenderer**
Renderer Class	**org.primefaces.component.imageswitch.ImageSwitchRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
effect	null	String	Name of the effect for transition.
speed	500	Integer	Speed of the effect in milliseconds.
slideshowSpeed	3000	Integer	Slideshow speed in milliseconds.
slideshowAuto	TRUE	Boolean	Starts slideshow automatically on page load.
style	null	String	Style of the main container.
styleClass	null	String	Style class of the main container.

Getting Started with ImageSwitch

ImageSwitch component needs a set of images to display. Provide the image collection as a set of children components.

```
<p:imageSwitch effect="FlyIn" widgetVar="imageswitch">
      <p:graphicImage value="/images/nature1.jpg" />
      <p:graphicImage value="/images/nature2.jpg" />
      <p:graphicImage value="/images/nature3.jpg" />
      <p:graphicImage value="/images/nature4.jpg" />
</p:imageSwitch>
```

Most of the time, images could be dynamic, ui:repeat is supported to implement this case.

```
<p:imageSwitch widgetVar="imageswitch">
      <ui:repeat value="#{bean.images}" var="image">
            <p:graphicImage value="#{image}" />
      </ui:repeat>
</p:imageSwitch>
```

Slideshow or Manual

ImageSwitch is in slideShow mode by default, if you'd like manual transitions disable slideshow and use client side api to create controls.

```
<p:imageSwitch effect="FlyIn" widgetVar="imageswitch">
      //images
</p:imageSwitch>

<span onclick="imageswitch.previous();">Previous</span>
<span onclick="imageswitch.next();">Next</span>
```

Client Side API

Method	Description
void previous()	Switches to previous image.
void next()	Switches to next image.
void startSlideshow()	Starts slideshow mode.
void stopSlideshow()	Stops slideshow mode.
voud pauseSlideshow();	Pauses slideshow mode.
void toggleSlideShow()	Toggles slideshow mode.

Effect Speed

The speed is considered in terms of milliseconds and specified via the speed attribute.

```
<p:imageSwitch effect="FlipOut" speed="150" widgetVar="imageswitch" >
      //set of images
</p:imageSwitch>
```

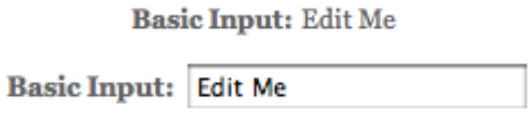
List of Effects

ImageSwitch supports a wide range of transition effects. Following is the full list, note that values are case sensitive.

- blindX
- blindY
- blindZ
- cover
- curtainX
- curtainY
- fade
- fadeZoom
- growX
- growY
- none
- scrollUp
- scrollDown
- scrollLeft
- scrollRight
- scrollVert
- shuffle
- slideX
- slideY
- toss
- turnUp
- turnDown
- turnLeft
- turnRight
- uncover
- wipe
- zoom

3.48 Inplace

Inplace provides easy inplace editing and inline content display. Inplace consists of two members, display element is the initial clickable label and inline element is the hidden content that is displayed when display element is toggled.



Info

Tag	**inplace**
Component Class	**org.primefaces.component.inplace.Inplace**
Component Type	**org.primefaces.component.Inplace**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.InplaceRenderer**
Renderer Class	**org.primefaces.component.inplace.InplaceRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
label	null	String	Label to be shown in display mode.
emptyLabel	null	String	Label to be shown in display mode when value is empty.
effect	fade	String	Effect to be used when toggling.
effectSpeed	normal	String	Speed of the effect.
disabled	FALSE	Boolean	Prevents hidden content to be shown.
style	null	String	Inline style of the main container element.
styleClass	null	String	Style class of the main container element.
editor	FALSE	Boolean	Specifies the editor mode.

Name	Default	Type	Description
saveLabel	Save	String	Tooltip text of save buttin in editor mode.
cancelLabel	Cancel	String	Tooltip text of cancel buttin in editor mode.
event	click	String	Name of the client side event to display inline content.
toggleable	TRUE	Boolean	Defines if inplace is toggleable or not.

Getting Started with Inplace

The inline component needs to be a child of inplace.

```
<p:inplace>
  <h:inputText value="Edit me" />
</p:inplace>
```

Custom Labels

By default inplace displays it's first child's value as the label, you can customize it via the label attribute.

```
<h:outputText value="Select One:" />

<p:inplace label="Cities">
      <h:selectOneMenu>
            <f:selectItem itemLabel="Istanbul" itemValue="Istanbul" />
            <f:selectItem itemLabel="Ankara" itemValue="Ankara" />
      </h:selectOneMenu>
</p:inplace>
```

Effects

Default effect is *fade* and other possible effect is *slide*, also effect speed can be tuned with values *slow*, *normal* and *fast*.

```
<p:inplace label="Show Image" effect="slide" effectSpeed="fast">
      <p:graphicImage value="/images/nature1.jpg" />
</p:inplace>
```

Ajax Behavior Events

Inplace editing is enabled via *editor* option.

```
public class InplaceBean {

      private String text;

      //getter-setter
}
```

```
<p:inplace editor="true">
      <h:inputText value="#{inplaceBean.text}" />
</p:inplace>
```

save and *cancel* are two provided ajax behaviors events you can use to hook-in the editing process.

```
public class InplaceBean {

      private String text;

      public void handleSave() {
            //add faces message with update text value
      }

      //getter-setter
}
```

```
<p:inplace editor="true">
      <p:ajax event="save" listener="#{inplaceBean.handleSave}" update="msgs" />
      <h:inputText value="#{inplaceBean.text}" />
</p:inplace>

<p:growl id="msgs" />
```

Client Side API

Widget: *PrimeFaces.widget.Inplace*

Method	Params	Return Type	Description
show()	-	void	Shows content and hides display element.
hide()	-	void	Shows display element and hides content.

Method	Params	Return Type	Description
toggle()	-	void	Toggles visibility of between content and display element.
save()	-	void	Triggers an ajax request to process inplace input.
cancel()	-	void	Triggers an ajax request to revert inplace input.

Skinning

Inplace resides in a main container element which *style* and *styleClass* options apply.

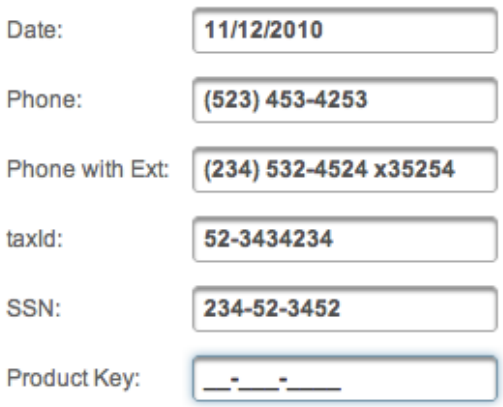
Following is the list of structural style classes;

Style Class	Applies
.ui-inplace	Main container element.
.ui-inplace-disabled	Main container element when disabled.
.ui-inplace-display	Display element.
.ui-inplace-content	Inline content.
.ui-inplace-editor	Editor controls container.
.ui-inplace-save	Save button.
.ui-inplace-cancel	Cancel button.

As skinning style classes are global, see the main Skinning section for more information.

3.49 InputMask

InputMask forces an input to fit in a defined mask template.



Info

Tag	**inputMask**
Component Class	**org.primefaces.component.inputmask.InputMask**
Component Type	**org.primefaces.component.InputMask**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.InputMaskRenderer**
Renderer Class	**org.primefaces.component.inputmask.InputMaskRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
mask	null	String	Mask template
placeHolder	null	String	PlaceHolder in mask template.
value	null	Object	Value of the component than can be either an EL expression of a literal text

Name	Default	Type	Description
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.

Name	Default	Type	Description
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Getting Started with InputMask

InputMask below enforces input to be in 99/99/9999 date format.

```
<p:inputMask value="#{bean.field}" mask="99/99/9999" />
```

Mask Samples

Here are more samples based on different masks;

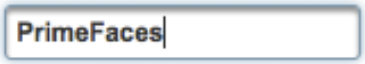
```
 <h:outputText value="Phone: " />
<p:inputMask value="#{bean.phone}" mask="(999) 999-9999"/>

<h:outputText value="Phone with Ext: " />
<p:inputMask value="#{bean.phoneExt}" mask="(999) 999-9999? x99999"/>

<h:outputText value="SSN: " />
<p:inputMask value="#{bean.ssn}" mask="999-99-9999"/>

<h:outputText value="Product Key: " />
<p:inputMask value="#{bean.productKey}" mask="a*-999-a999"/>
```

Skinning

style and *styleClass* options apply to the displayed input element. As skinning style classes are global, see the main Skinning section for more information.

3.50 InputText

InputText is an extension to standard inputText with skinning capabilities.



Info

Tag	**inputText**
Component Class	**org.primefaces.component.inputtext.InputText**
Component Type	**org.primefaces.component.InputText**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.InputTextRenderer**
Renderer Class	**org.primefaces.component.inputtext.InputTextRender**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	Method Expr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	Method Expr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.

Name	Default	Type	Description
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.

Name	Default	Type	Description
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.
type	text	String	Input field type.

Getting Started with InputText

InputText usage is same as standard inputText;

```
<p:inputText value="#{bean.propertyName}" />
```

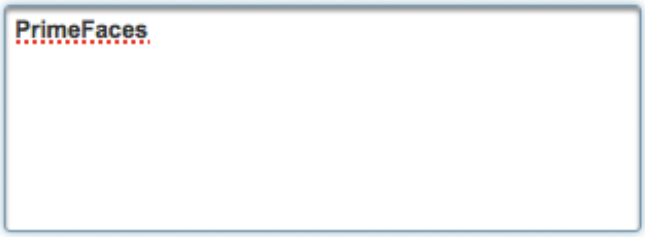
```
public class Bean {

      private String propertyName;

      //getter and setter
}
```

Skinning

style and *styleClass* options apply to the input element. As skinning style classes are global, see the main Skinning section for more information.

3.51 InputTextarea

InputTextarea is an extension to standard inputTextara with skinning capabilities and auto growing.



Info

Tag	**inputTextarea**
Component Class	**org.primefaces.component.inputtextarea.InputTextarea**
Component Type	**org.primefaces.component.InputTextarea**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.InputTextareaRenderer**
Renderer Class	**org.primefaces.component.inputtextarea.InputTextareaRender**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required

Name	Default	Type	Description
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
autoResize	TRUE	Boolean	Specifies auto growing when being typed.
counter	null	String	Id of the output component to display remaining chars.
counterTemplate	{0}	String	Template text to display in counter.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.

Name	Default	Type	Description
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Getting Started with InputTextarea

InputTextarea usage is same as standard inputTextarea;

```
<p:inputTextarea value="#{bean.propertyName}" />
```

AutoResize

When textarea is being typed, if content height exceeds the allocated space, textarea can grow automatically. Use autoResize option to turn on/off this feature.

```
<p:inputTextarea value="#{bean.propertyName}" autoResize="true|false"/>
```

Remaining Characters

InputTextarea can limit the maximum allowed characters with maxLength option and display the remaining characters count as well.

```
<p:inputTextarea value="#{bean.propertyName}" counter="display"
            maxlength="20" counterTemplate="{0} characters remaining" />

<h:outputText id="display" />
```

Skinning

style and *styleClass* options apply to the textarea element along with the default *ui-inputtextarea* style class. As skinning style classes are global, see the main Skinning section for more information.

3.52 Keyboard

Keyboard is an input component that uses a virtual keyboard to provide the input. Notable features are the customizable layouts and skinning capabilities.

Info

Tag	**keyboard**
Component Class	**org.primefaces.component.keyboard.Keyboard**
Component Type	**org.primefaces.component.Keyboard**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.KeyboardRenderer**
Renderer Class	**org.primefaces.component.keyboard.KeyboardRenderer**

Attributes

Name	Default	Type	Description
id	Assigned by JSF	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.

Name	Default	Type	Description
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
password	FALSE	Boolean	Makes the input a password field.
showMode	focus	String	Specifies the showMode, 'focus', 'button', 'both'
buttonImage	null	String	Image for the button.
buttonImageOnly	FALSE	boolean	When set to true only image of the button would be displayed.
effect	fadeIn	String	Effect of the display animation.
effectDuration	null	String	Length of the display animation.
layout	qwerty	String	Built-in layout of the keyboard.
layoutTemplate	null	String	Template of the custom layout.
keypadOnly	focus	Boolean	Specifies displaying a keypad instead of a keyboard.
promptLabel	null	String	Label of the prompt text.
closeLabel	null	String	Label of the close key.
clearLabel	null	String	Label of the clear key.
backspaceLabel	null	String	Label of the backspace key.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.

Name	Default	Type	Description
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Name	Default	Type	Description
widgetVar	null	String	Name of the client side widget.

Getting Started with Keyboard

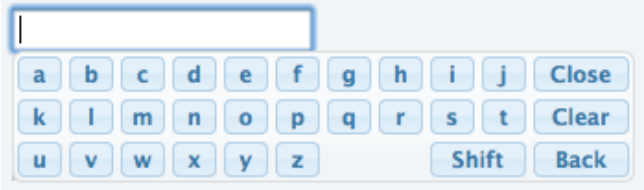
Keyboard is used just like a simple inputText, by default when the input gets the focus a keyboard is displayed.

```
<p:keyboard value="#{bean.value}" />
```

Built-in Layouts

There're a couple of built-in keyboard layouts these are 'qwerty', 'qwertyBasic' and 'alphabetic'. For example keyboard below has the alphabetic layout.

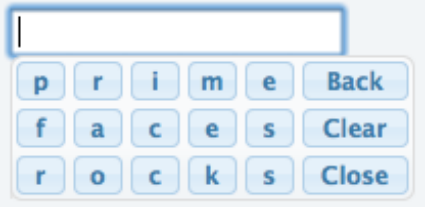
```
<p:keyboard value="#{bean.value}" layout="alphabetic"/>
```



Custom Layouts

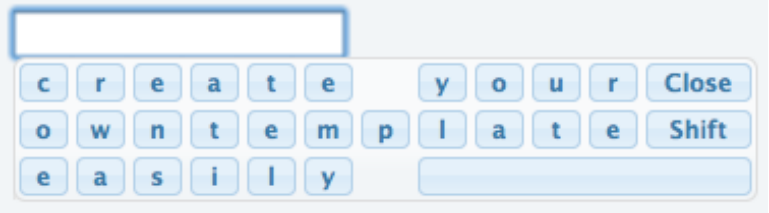
Keyboard has a very flexible layout mechanism allowing you to come up with your own layout.

```
<p:keyboard value="#{bean.value}"
      layout="custom"
      layoutTemplate="prime-back,faces-clear,rocks-close"/>
```



Another example;

```
<p:keyboard value="#{bean.value}"
      layout="custom"
      layoutTemplate="create-space-your-close,owntemplate-shift,easily-space-
spacebar"/>
```



A layout template basically consists of built-in keys and your own keys. Following is the list of all built-in keys.

- back
- clear
- close
- shift
- spacebar
- space
- halfspace

All other text in a layout is realized as seperate keys so "prime" would create 5 keys as "p" "r" "i" "m" "e". Use dash to seperate each member in layout and use commas to create a new row.

Keypad

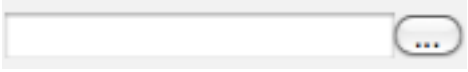
By default keyboard displays whole keys, if you only need the numbers use the keypad mode.

```
<p:keyboard value="#{bean.value}" keypadOnly="true"/>
```

ShowMode

There're a couple of different ways to display the keyboard, by default keyboard is shown once input field receives the focus. This is customized using the showMode feature which accept values 'focus', 'button', 'both'. Keyboard below displays a button next to the input field, when the button is clicked the keyboard is shown.

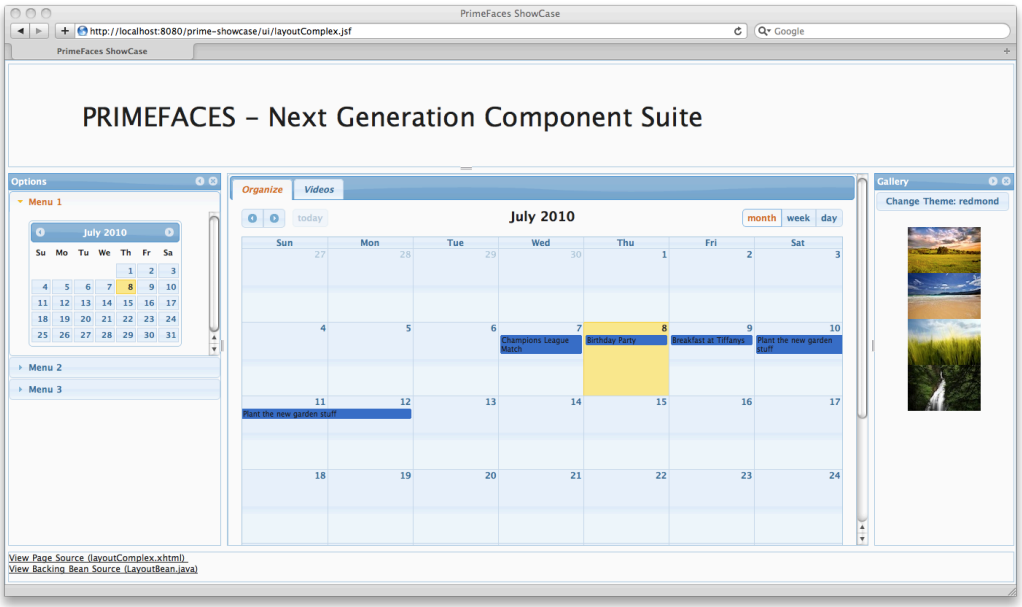
```
<p:keyboard value="#{bean.value}" showMode="button"/>
```



Button can also be customized using the *buttonImage* and *buttonImageOnly* attributes.

3.53 Layout

Layout component features a highly customizable borderLayout model making it very easy to create complex layouts even if you're not familiar with web design.



Info

Tag	**layout**
Component Class	**org.primefaces.component.layout.Layout**
Component Type	**org.primefaces.component.Layout**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.LayoutRenderer**
Renderer Class	**org.primefaces.component.layout.LayoutRenderer**

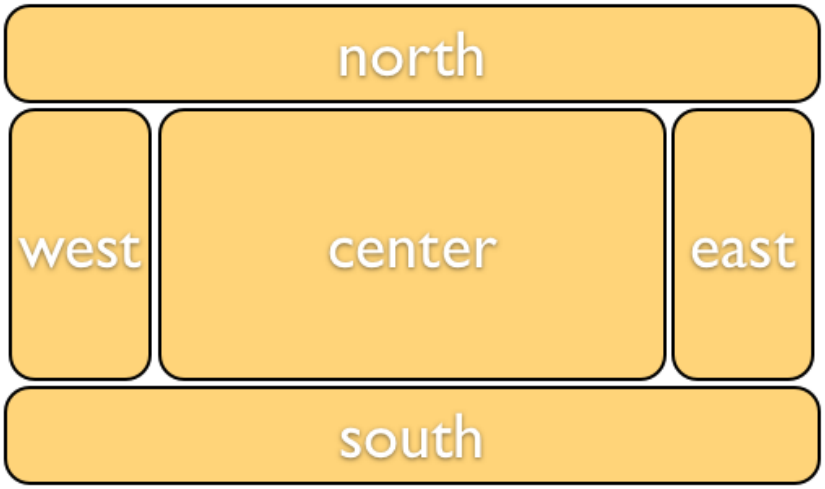
Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
fullPage	FALSE	Boolean	Specifies whether layout should span all page or not.

Name	Default	Type	Description
style	null	String	Style to apply to container element, this is only applicable to element based layouts.
styleClass	null	String	Style class to apply to container element, this is only applicable to element based layouts.
onResize	null	String	Client side callback to execute when a layout unit is resized.
onClose	null	String	Client side callback to execute when a layout unit is closed.
onToggle	null	String	Client side callback to execute when a layout unit is toggled.
resizeTitle	null	String	Title label of the resize button.
collapseTitle	null	String	Title label of the collapse button.
expandTitle	null	String	Title label of the expand button.
closeTitle	null	String	Title label of the close button.

Getting started with Layout

Layout is based on a borderLayout model that consists of 5 different layout units which are top, left, center, right and bottom. This model is visualized in the schema below;



Full Page Layout

Layout has two modes, you can either use it for a full page layout or for a specific region in your page. This setting is controlled with the fullPage attribute which is false by default.

The regions in a layout are defined by layoutUnits, following is a simple full page layout with all possible units. Note that you can place any content in each layout unit.

```
<p:layout fullPage="true">
      <p:layoutUnit position="north" size="50">
            <h:outputText value="Top content." />
      </p:layoutUnit>
      <p:layoutUnit position="south" size="100">
            <h:outputText value="Bottom content." />
      </p:layoutUnit>
      <p:layoutUnit position="west" size="300">
            <h:outputText value="Left content" />
      </p:layoutUnit>
      <p:layoutUnit position="east" size="200">
            <h:outputText value="Right Content" />
      </p:layoutUnit>
      <p:layoutUnit position="center">
            <h:outputText value="Center Content" />
      </p:layoutUnit>
</p:layout>
```

Forms in Full Page Layout

When working with forms and full page layout, avoid using a form that contains layoutunits as generated dom may not be the same. So following is **invalid**.

```
<p:layout fullPage="true">

      <h:form>

            <p:layoutUnit position="west" size="100">
                  h:outputText value="Left Pane" />
            </p:layoutUnit>

            <p:layoutUnit position="center">
                  <h:outputText value="Right Pane" />
            </p:layoutUnit>

      </h:form>

</p:layout>
```

A layout unit must have it's own form instead, also avoid trying to update layout units because of same reason, update it's content instead.

Dimensions

Except center layoutUnit, other layout units **must** have dimensions defined via *size* option.

Element based layout

Another use case of layout is the element based layout. This is the default case actually so just ignore fullPage attribute or set it to false. Layout example below demonstrates creating a split panel implementation.

```
<p:layout style="width:400px;height:200px">

      <p:layoutUnit position="west" size="100">
            <h:outputText value="Left Pane" />
      </p:layoutUnit>

      <p:layoutUnit position="center">
            <h:outputText value="Right Pane" />
      </p:layoutUnit>

      //more layout units

</p:layout>
```

Ajax Behavior Events

Layout provides custom ajax behavior events for each layout state change.

Event	Listener Parameter	Fired
toggle	org.primefaces.event.ToggleEvent	When a unit is expanded or collapsed.
close	org.primefaces.event.CloseEvent	When a unit is closed.
resize	org.primefaces.event.ResizeEvent	When a unit is resized.

Stateful Layout

Making layout stateful would be easy, once you create your data to store the user preference, you can update this data using ajax event listeners provided by layout. For example if a layout unit is collapsed, you can save and persist this information. By binding this persisted information to the collapsed attribute of the layout unit layout will be rendered as the user left it last time.

Client Side API

Widget: *PrimeFaces.widget.Layout*

Method	Params	Return Type	Description
toggle(position)	position	void	Toggles layout unit.
show(position)	position	void	Shows layout unit.
hide(unit)	position	void	Hides layout unit.

Skinning

Following is the list of structural style classes;

Style Class	Applies
.ui-layout	Main wrapper container element
.ui-layout-doc	Layout container
.ui-layout-unit	Each layout unit container
.ui-layout-{position}	Position based layout unit
.ui-layout-unit-header	Layout unit header
.ui-layout-unit-content	Layout unit body

As skinning style classes are global, see the main Skinning section for more information.

3.54 LayoutUnit

LayoutUnit represents a region in the border layout model of the Layout component.

Info

Tag	**layoutUnit**
Component Class	**org.primefaces.component.layout.LayoutUnit**
Component Type	**org.primefaces.component.LayoutUnit**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
position	null	String	Position of the unit.
size	null	String	Size of the unit.
resizable	FALSE	Boolean	Makes the unit resizable.
closable	FALSE	Boolean	Makes the unit closable.
collapsible	FALSE	Boolean	Makes the unit collapsible.
header	null	String	Text of header.
footer	null	String	Text of footer.
minSize	null	Integer	Minimum dimension for resize.
maxSize	null	Integer	Maximum dimension for resize.
gutter	4px	String	Gutter size of layout unit.
visible	TRUE	Boolean	Specifies default visibility
collapsed	FALSE	Boolean	Specifies toggle status of unit
collapseSize	null	Integer	Size of the unit when collapsed
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.

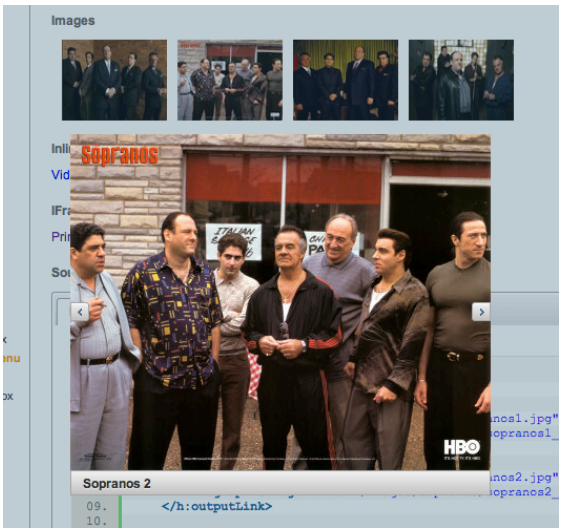
Name	Default	Type	Description
effect	null	String	Effect name of the layout transition.
effectSpeed	null	String	Effect speed of the layout transition.

Getting started with LayoutUnit

See layout component documentation for more information regarding the usage of layoutUnits.

3.55 LightBox

Lightbox features a powerful overlay that can display images, multimedia content, other JSF components and external urls.



Info

Tag	**lightBox**
Component Class	**org.primefaces.component lightbox.LightBox**
Component Type	**org.primefaces.component.LightBox**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.LightBoxRenderer**
Renderer Class	**org.primefaces.component.lightbox.LightBoxRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
style	null	String	Style of the container element not the overlay element.
styleClass	null	String	Style class of the container element not the overlay element.

Name	Default	Type	Description
width	null	String	Width of the overlay in iframe mode.
heigth	null	String	Height of the overlay in iframe mode.
iframe	FALSE	Boolean	Specifies an iframe to display an external url in overlay.
visible	FALSE	Boolean	Displays lightbox without requiring any user interaction by default.
onHide	null	String	Client side callback to execute when lightbox is displayed.
onShow	null	String	Client side callback to execute when lightbox is hidden.

Images

The images displayed in the lightBox need to be nested as child outputLink components. Following lightBox is displayed when any of the links are clicked.

```
<p:lightBox>
      <h:outputLink value="sopranos/sopranos1.jpg" title="Sopranos 1">
            <h:graphicImage value="sopranos/sopranos1_small.jpg/>
      </h:outputLink>

      //more
</p:lightBox>
```

IFrame Mode

LightBox also has the ability to display iframes inside the page overlay, following lightbox displays the PrimeFaces homepage when the link inside is clicked.

```
<p:lightBox iframe="true">
      <h:outputLink value="http://www.primefaces.org" title="PrimeFaces HomePage">
            <h:outputText value="PrimeFaces HomePage"/>
      </h:outputLink>
</p:lightBox>
```

Clicking the outputLink will display PrimeFaces homepage within an iframe.

Inline Mode

Inline mode acts like a modal dialog, you can display other JSF content on the page using the lightbox overlay. Simply place your overlay content in the "inline" facet. Clicking the link in the example below will display the panelGrid contents in overlay.

```
<p:lightBox>
      <h:outputLink value="#" title="Leo Messi" >
            <h:outputText value="The Messiah"/>
      </h:outputLink>

      <f:facet name="inline">
            //content here
      </f:facet>
</p:lightBox>
```

Inline Mode

Inline mode acts like a modal dialog, you can display other JSF content on the page using the lightbox overlay. Simply place your overlay content in the "inline" facet. Clicking the link in the example below will display the panelGrid contents in overlay.

Client Side API

Widget: *PrimeFaces.widget.LightBox*

Method	Params	Return Type	Description
show()	-	void	Displays lightbox.
hide()	-	void	Hides lightbox.

Skinning

Lightbox resides in a main container element which *style* and *styleClass* options apply.

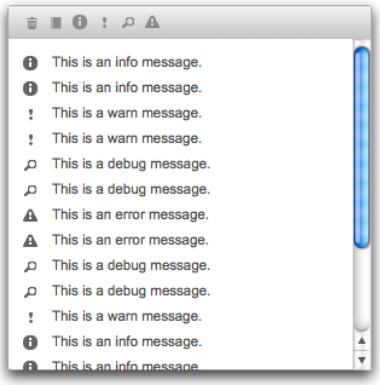
Following is the list of structural style classes;

Style Class	Applies
.ui-lightbox	Main container element.
.ui-lightbox-content-wrapper	Content wrapper element.
.ui-lightbox-content	Content container.
.ui-lightbox-nav-right	Next image navigator.
.ui-lightbox-nav-left	Previous image navigator.
.ui-lightbox-loading	Loading image.
.ui-lightbox-caption	Caption element.

As skinning style classes are global, see the main Skinning section for more information.

3.56 Log

Log component is a visual console to display logs on JSF pages.


This is an info message.
This is an info message.
This is a warn message.
This is a warn message.
This is a debug message.
This is a debug message.
This is an error message.
This is an error message.
This is a debug message.
This is a debug message.
This is a warn message.
This is an info message.

Info

Tag	**log**
Component Class	**org.primefaces.component.log.Log**
Component Type	**org.primefaces.component.Log**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.LogRenderer**
Renderer Class	**org.primefaces.component.log.LogRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting started with Log

Log component is used simply as adding the component to the page.

```
<p:log />
```

Log API

PrimeFaces uses client side log apis internally, for example you can use log component to see details of an ajax request. Log API is also available via global PrimeFaces object in case you'd like to use the log component to display your logs.

```
<script type="text/javascript">
      PrimeFaces.info('Info message');
      PrimeFaces.debug('Debug message');
      PrimeFaces.warn('Warning message');
      PrimeFaces.error('Error message');
 </script>
```

3.57 Media

Media component is used for embedding multimedia content such as videos and music to JSF views. Media renders <object /> or <embed /> html tags depending on the user client.

Info

Tag	**media**
Component Class	**org.primefaces.component.media.Media**
Component Type	**org.primefaces.component.Media**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MediaRenderer**
Renderer Class	**org.primefaces.component.media.MediaRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	String	Media source to play.
player	null	String	Type of the player, possible values are "quicktime","windows","flash","real".
width	null	String	Width of the player.
height	null	String	Height of the player.
style	null	String	Style of the player.
styleClass	null	String	StyleClass of the player.

Getting started with Media

In it's simplest form media component requires a source to play, this is defined using the value attribute.

```
<p:media value="/media/ria_with_primefaces.mov" />
```

Player Types

By default, players are identified using the value extension so for instance mov files will be played by quicktime player. You can customize which player to use with the player attribute.

```
<p:media value="http://www.youtube.com/v/ABCDEFGH" player="flash"/>
```

Following is the supported players and file types.

Player	Types
windows	asx, asf, avi, wma, wmv
quicktime	aif, aiff, aac, au, bmp, gsm, mov, mid, midi, mpg, mpeg, mp4, m4a, psd, qt, qtif, qif, qti, snd, tif, tiff, wav, 3g2, 3pg
flash	flv, mp3, swf
real	ra, ram, rm, rpm, rv, smi, smil

Parameters

Different proprietary players might have different configuration parameters, these can be specified using f:param tags.

```
<p:media value="/media/ria_with_primefaces.mov">
     <f:param name="param1" value="value1" />
     <f:param name="param2" value="value2" />
 </p:media>
```

StreamedContent Support

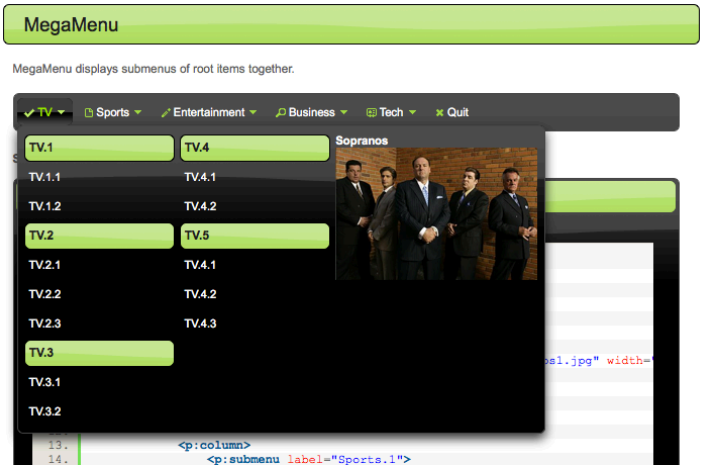
Media component can also play binary media content, example for this use case is storing media files in datatabase using binary format. In order to implement this, bind a StreamedContent.

```
<p:media value="#{mediaBean.media}" width="250" height="225" player="quicktime"/>
```

```
public class MediaBean {

      private StreamedContent media;

      public MediaController() {
            InputStream stream = //Create binary stream from database
            media = new DefaultStreamedContent(stream, "video/quicktime");
      }
      public StreamedContent getMedia() { return media; }
}
```

3.58 MegaMenu

MegaMenu is a horizontal navigation component that displays submenus together.



Info

Tag	**megaMenu**
Component Class	**org.primefaces.component.megamenu.MegaMenu**
Component Type	**org.primefaces.component.MegaMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MegaMenuRenderer**
Renderer Class	**org.primefaces.component.megamenu.MegaMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget
model	null	MenuModel	MenuModel instance to create menus programmatically
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.

Name	Default	Type	Description
autoDisplay	TRUE	Boolean	Defines whether submenus will be displayed on mouseover or not. When set to false, click event is required to display.

Getting Started with MegaMenu

Layout of MegaMenu is grid based and root items require columns as children to define each section in a grid.

```
<p:megaMenu>
      <p:submenu label="TV" icon="ui-icon-check">
            <p:column>
                  <p:submenu label="TV.1">
                        <p:menuitem value="TV.1.1" url="#" />
                        <p:menuitem value="TV.1.2" url="#" />
                  </p:submenu>
                  <p:submenu label="TV.2">
                        <p:menuitem value="TV.2.1" url="#" />
                        <p:menuitem value="TV.2.2" url="#" />
                        <p:menuitem value="TV.2.3" url="#" />
                  </p:submenu>
                  <p:submenu label="TV.3">
                        <p:menuitem value="TV.3.1" url="#" />
                        <p:menuitem value="TV.3.2" url="#" />
                  </p:submenu>
            </p:column>

            <p:column>
                  <p:submenu label="TV.4">
                        <p:menuitem value="TV.4.1" url="#" />
                        <p:menuitem value="TV.4.2" url="#" />
                  </p:submenu>
                  <p:submenu label="TV.5">
                        <p:menuitem value="TV.5.1" url="#" />
                        <p:menuitem value="TV.5.2" url="#" />
                        <p:menuitem value="TV.5.3" url="#" />
                  </p:submenu>
                  <p:submenu label="TV.6">
                        <p:menuitem value="TV.6.1" url="#" />
                        <p:menuitem value="TV.6.2" url="#" />
                        <p:menuitem value="TV.6.3" url="#" />
                  </p:submenu>

            </p:column>

      </p:submenu>

      //more root items
</p:megaMenu>
```

Custom Content

Any content can be placed inside columns.

```
<p:column>
      <strong>Sopranos</strong>
      <p:graphicImage value="/images/sopranos/sopranos1.jpg" width="200"/>
</p:column>
```

Root Menuitem

MegaMenu supports menuitem as root menu options as well. Following example allows a root menubar item to execute an action to logout the user.

```
<p:megaMenu>
      //submenus
      <p:menuitem label="Logout" action="#{bean.logout}"/>
</p:megaMenu>
```

Dynamic Menus

Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

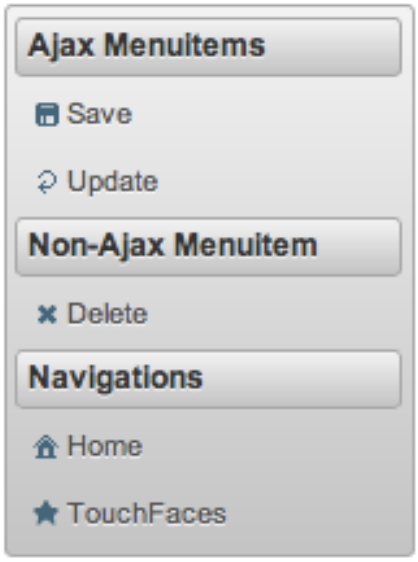
MegaMenu resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-megamenu	Container element of menubar.
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.59 Menu

Menu is a navigation component with various customized modes like multi tiers, ipod style sliding and overlays.



Info

Tag	**menu**
Component Class	**org.primefaces.component.menu.Menu**
Component Type	**org.primefaces.component.Menu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MenuRenderer**
Renderer Class	**org.primefaces.component.menu.MenuRenderer**

Attributes

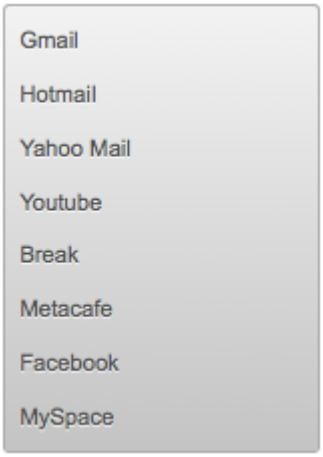
Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget.
model	null	MenuModel	A menu model instance to create menu programmatically.

Name	Default	Type	Description
trigger	null	String	Id of component whose click event will show the dynamic positioned menu.
my	null	String	Corner of menu to align with trigger element.
at	null	String	Corner of trigger to align with menu element.
overlay	FALSE	Boolean	Defines positioning type of menu, either static or overlay.
style	null	String	Inline style of the main container element.
styleClass	null	String	Style class of the main container element.
triggerEvent	click	String	Event to show the dynamic positioned menu.

Getting started with the Menu

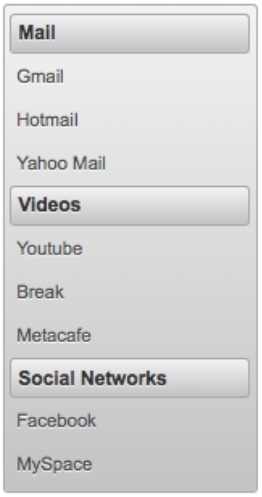
A menu is composed of submenus and menuitems.

```
<p:menu>
      <p:menuitem value="Gmail" url="http://www.google.com" />
      <p:menuitem value="Hotmail" url="http://www.hotmail.com" />
      <p:menuitem value="Yahoo Mail" url="http://mail.yahoo.com" />
      <p:menuitem value="Youtube" url="http://www.youtube.com" />
      <p:menuitem value="Break" url="http://www.break.com" />
      <p:menuitem value="Metacafe" url="http://www.metacafe.com" />
      <p:menuitem value="Facebook" url="http://www.facebook.com" />
      <p:menuitem value="MySpace" url="http://www.myspace.com" />
</p:menu>
```



Submenus are used to group menuitems;

```
<p:menu>
      <p:submenu label="Mail">
            <p:menuitem value="Gmail" url="http://www.google.com" />
            <p:menuitem value="Hotmail" url="http://www.hotmail.com" />
            <p:menuitem value="Yahoo Mail" url="http://mail.yahoo.com" />
      </p:submenu>

      <p:submenu label="Videos">
            <p:menuitem value="Youtube" url="http://www.youtube.com" />
            <p:menuitem value="Break" url="http://www.break.com" />
            <p:menuitem value="Metacafe" url="http://www.metacafe.com" />
      </p:submenu>

      <p:submenu label="Social Networks">
            <p:menuitem value="Facebook" url="http://www.facebook.com" />
            <p:menuitem value="MySpace" url="http://www.myspace.com" />
      </p:submenu>
</p:menu>
```



Overlay Menu

Menu can be positioned on a page in two ways; "static" and "dynamic". By default it's static meaning the menu is in normal page flow. In contrast dynamic menus is not on the normal flow of the page allowing them to overlay other elements.

A dynamic menu is created by setting *overlay* option to true and defining a trigger to show the menu. Location of menu on page will be relative to the trigger and defined by my and at options that take combination of four values;

- left
- right
- bottom
- top

For example, clicking the button below will display the menu whose top left corner is aligned with bottom left corner of button.

```
<p:menu overlay="true" trigger="btn" my="left top" at="bottom left">
      ...submenus and menuitems
</p:menu>

<p:commandButton id="btn" value="Show Menu" type="button"/>
```

Ajax and Non-Ajax Actions

As menu uses menuitems, it is easy to invoke actions with or without ajax as well as navigation. See menuitem documentation for more information about the capabilities.

```
<p:menu>
  <p:submenu label="Options">
    <p:menuitem value="Save" actionListener="#{bean.save}" update="comp"/>
    <p:menuitem value="Update" actionListener="#{bean.update}" ajax="false"/>
    <p:menuitem value="Navigate" url="http://www.primefaces.org"/>
  </p:submenu>
</p:menu>
```

Dynamic Menus

Menus can be created programmatically as well, this is more flexible compared to the declarative approach. Menu metadata is defined using an *org.primefaces.model.MenuModel* instance, PrimeFaces provides the built-in *org.primefaces.model.DefaultMenuModel* implementation. For further customization you can also create and bind your own MenuModel implementation.

```
public class MenuBean {

      private MenuModel model;

      public MenuBean() {
            model = new DefaultMenuModel();

            //First submenu
            Submenu submenu = new Submenu();
            submenu.setLabel("Dynamic Submenu 1");

            MenuItem item = new MenuItem();
            item.setValue("Dynamic Menuitem 1.1");
            item.setUrl("#");
            submenu.getChildren().add(item);

            model.addSubmenu(submenu);
      }

      public MenuModel getModel() { return model; }
}
```

```
<p:menu model="#{menuBean.model}" />
```

Skinning

Menu resides in a main container element which *style* and *styleClass* attributes apply.

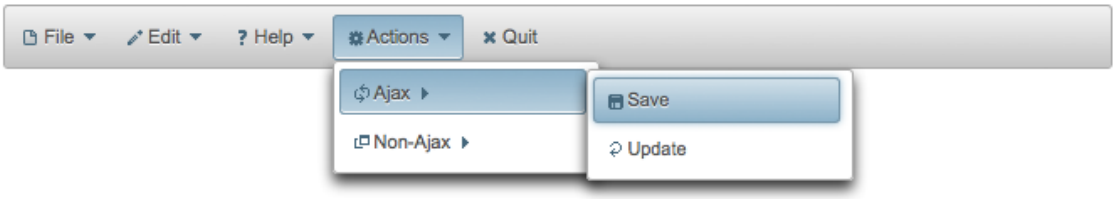
Following is the list of structural style classes;

Style Class	Applies
.ui-menu	Container element of menu
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item
.ui-menu-sliding	Container of ipod like sliding menu

As skinning style classes are global, see the main Skinning section for more information.

3.60 Menubar

Menubar is a horizontal navigation component.



Info

Tag	**menubar**
Component Class	**org.primefaces.component.menubar.Menubar**
Component Type	**org.primefaces.component.Menubar**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MenubarRenderer**
Renderer Class	**org.primefaces.component.menubar.MenubarRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget
model	null	MenuModel	MenuModel instance to create menus programmatically
style	null	String	Inline style of menubar
styleClass	null	String	Style class of menubar
autoDisplay	FALSE	Boolean	Defines whether the first level of submenus will be displayed on mouseover or not. When set to false, click event is required to display.

Getting started with Menubar

Submenus and menuitems as child components are required to compose the menubar.

```
<p:menubar>
      <p:submenu label="Mail">
            <p:menuitem value="Gmail" url="http://www.google.com" />
            <p:menuitem value="Hotmail" url="http://www.hotmail.com" />
            <p:menuitem value="Yahoo Mail" url="http://mail.yahoo.com" />
      </p:submenu>
      <p:submenu label="Videos">
            <p:menuitem value="Youtube" url="http://www.youtube.com" />
            <p:menuitem value="Break" url="http://www.break.com" />
      </p:submenu>
</p:menubar>
```

Nested Menus

To create a menubar with a higher depth, nest submenus in parent submenus.

```
<p:menubar>
      <p:submenu label="File">
            <p:submenu label="New">
                  <p:menuitem value="Project" url="#"/>
                  <p:menuitem value="Other" url="#"/>
            </p:submenu>
            <p:menuitem value="Open" url="#"></p:menuitem>
            <p:menuitem value="Quit" url="#"></p:menuitem>
      </p:submenu>
      <p:submenu label="Edit">
            <p:menuitem value="Undo" url="#"></p:menuitem>
            <p:menuitem value="Redo" url="#"></p:menuitem>
      </p:submenu>
      <p:submenu label="Help">
            <p:menuitem label="Contents" url="#" />
            <p:submenu label="Search">
                  <p:submenu label="Text">
                        <p:menuitem value="Workspace" url="#" />
                  </p:submenu>
                  <p:menuitem value="File" url="#" />
            </p:submenu>
      </p:submenu>
</p:menubar>
```

Root Menuitem

Menubar supports menuitem as root menu options as well;

```
<p:menubar>
      <p:menuitem label="Logout" action="#{bean.logout}"/>
</p:menubar>
```

Ajax and Non-Ajax Actions

As menu uses menuitems, it is easy to invoke actions with or without ajax as well as navigation. See menuitem documentation for more information about the capabilities.

```
<p:menubar>
  <p:submenu label="Options">
    <p:menuitem value="Save" actionListener="#{bean.save}" update="comp"/>
    <p:menuitem value="Update" actionListener="#{bean.update}" ajax="false"/>
    <p:menuitem value="Navigate" url="http://www.primefaces.org"/>
  </p:submenu>
</p:menubar>
```

Dynamic Menus

Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

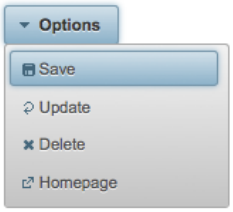
Menubar resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-menubar	Container element of menubar.
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.61 MenuButton

MenuButton displays different commands in a popup menu.



Info

Tag	**menuButton**
Component Class	**org.primefaces.component.menubutton.MenuButton**
Component Type	**org.primefaces.component.MenuButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MenuButtonRenderer**
Renderer Class	**org.primefaces.component.menubutton.MenuButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	String	Label of the button
style	null	String	Style of the main container element
styleClass	null	String	Style class of the main container element
widgetVar	null	String	Name of the client side widget
model	null	MenuModel	MenuModel instance to create menus programmatically
disabled	FALSE	Boolean	Disables or enables the button.

Getting started with the MenuButton

MenuButton consists of one ore more menuitems. Following menubutton example has three menuitems, first one is used triggers an action with ajax, second one does the similar but without ajax and third one is used for redirect purposes.

```
<p:menuButton value="Options">
      <p:menuitem value="Save" actionListener="#{bean.save}" update="comp" />
      <p:menuitem value="Update" actionListener="#{bean.update}" ajax="false" />
      <p:menuitem value="Go Home" url="/home.jsf" />
</p:menuButton>
```

Dynamic Menus

Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

MenuButton resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-menu	Container element of menu.
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item
.ui-button	Button element
.ui-button-text	Label of button

3.62 MenuItem

MenuItem is used by various menu components of PrimeFaces.

Info

Tag	**menuItem**
Tag Class	**org.primefaces.component.menuitem.MenuItemTag**
Component Class	**org.primefaces.component.menuitem.MenuItem**
Component Type	**org.primefaces.component.MenuItem**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
value	null	String	Label of the menuitem
actionListener	null	javax.el.MethodExpression	Action listener to be invoked when menuitem is clicked.
action	null	javax.el.MethodExpression	Action to be invoked when menuitem is clicked.
immediate	FALSE	Boolean	When true, action of this menuitem is processed after apply request phase.
url	null	String	Url to be navigated when menuitem is clicked
target	null	String	Target type of url navigation
style	null	String	Style of the menuitem label
styleClass	null	String	StyleClass of the menuitem label
onclick	null	String	Javascript event handler for click event
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component id(s) to process partially instead of whole view.

Name	Default	Type	Description
update	null	String	Client side id of the component(s) to be updated after async partial submit request.
disabled	FALSE	Boolean	Disables the menuitem.
onstart	null	String	Javascript handler to execute before ajax request is begins.
oncomplete	null	String	Javascript handler to execute when ajax request is completed.
onsuccess	null	String	Javascript handler to execute when ajax request succeeds.
onerror	null	String	Javascript handler to execute when ajax request fails.
global	TRUE	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
ajax	TRUE	Boolean	Specifies submit mode.
icon	null	String	Path of the menuitem image.
title	null	String	Advisory tooltip information.

Getting started with MenuItem

MenuItem is a generic component used by the following PrimeFaces components.

- Menu
- MenuBar
- Breadcrumb
- Dock
- Stack
- MenuButton

Note that some attributes of menuitem might not be supported by these menu components. Refer to the specific component documentation for more information.

Navigation vs Action

Menuitem has two use cases, directly navigating to a url with GET and doing a POST do execute an action which you can still do navigation with JSF navigation rules. This is decided by url attribute, if it is present menuitem does a GET request, if not parent form is posted.

Icons

There are two ways to specify an icon of a menuitem, you can either use bundled icons within PrimeFaces or provide your own via css.
ThemeRoller Icons

```
<p:menuitem icon="ui-icon-disk" ... />
```

Custom Icons

```
<p:menuitem icon="barca" ... />
```

```
.barca {
      background: url(barca_logo.png) no-repeat;
      height:16px;
      width:16px;
}
```

3.63 Message

Message is a pre-skinned extended version of the standard JSF message component.

Info

Tag	**message**
Component Class	**org.primefaces.component.message.Message**
Component Type	**org.primefaces.component.Message**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MessageRenderer**
Renderer Class	**org.primefaces.component.message.MessageRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
showSummary	FALSE	Boolean	Specifies if the summary of the FacesMessage should be displayed.
showDetail	TRUE	Boolean	Specifies if the detail of the FacesMessage should be displayed.
for	null	String	Id of the component whose messages to display.
redisplay	TRUE	Boolean	Defines if already rendered messages should be displayed
display	both	String	Defines the display mode.
escape	TRUE	Boolean	Defines whether html would be escaped or not.
severity	null	String	Comma separated list of severities to display only.

Getting started with Message

Message usage is exactly same as standard message.

```
<h:inputText id="txt" value="#{bean.text}" />
 <p:message for="txt" />
```

Display Mode

Message component has three different display modes;

- text : Only message text is displayed.
- icon : Only message severity is displayed and message text is visible as a tooltip.
- both (default) : Both icon and text are displayed.

Severity Levels

Using severity attribute, you can define which severities can be displayed by the component. For instance, you can configure messages to only display infos and warnings.

```
<p:message severity="info,warn" for="txt"/>
```

Escaping

Component escapes html content in messages by default, in case you need to display html, enable escape option.

```
<p:message escape="true" for="txt" />
```

Skinning Message

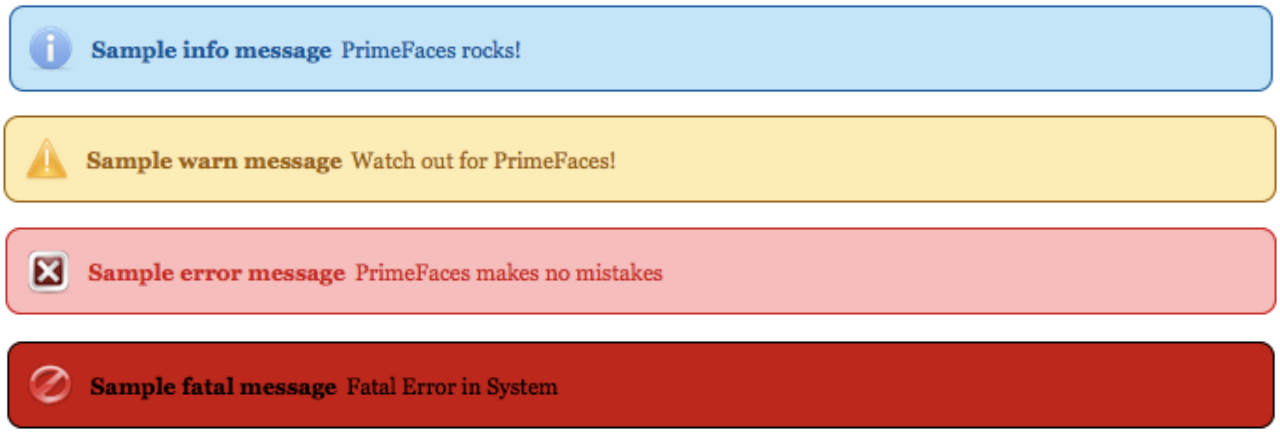
Full list of CSS selectors of message is as follows;

Style Class	Applies
ui-message-{severity}	Container element of the message
ui-message-{severity}-summary	Summary text
ui-message-{severity}-detail	Detail text

{severity} can be 'info', 'error', 'warn' and error.

3.64 Messages

Messages is a pre-skinned extended version of the standard JSF messages component.


Sample info message PrimeFaces rocks!
Sample warn message Watch out for PrimeFaces!
Sample error message PrimeFaces makes no mistakes
Sample fatal message Fatal Error in System

Info

Tag	**messages**
Component Class	**org.primefaces.component.messages.Messages**
Component Type	**org.primefaces.component.Messages**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.MessagesRenderer**
Renderer Class	**org.primefaces.component.messages.MessagesRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
showSummary	FALSE	Boolean	Specifies if the summary of the FacesMessages should be displayed.
showDetail	TRUE	Boolean	Specifies if the detail of the FacesMessages should be displayed.
globalOnly	FALSE	String	When true, only facesmessages with no clientIds are displayed.

Name	Default	Type	Description
redisplay	TRUE	Boolean	Defines if already rendered messages should be displayed
autoUpdate	FALSE	Boolean	Enables auto update mode if set true.
for	null	String	Name of associated key, takes precedence when used with globalOnly.
escape	TRUE	Boolean	Defines whether html would be escaped or not.
severity	null	String	Comma separated list of severities to display only.

Getting started with Messages

Message usage is exactly same as standard messages.

```
<p:messages />
```

AutoUpdate

When auto update is enabled, messages component is updated with each ajax request automatically.

Targetable Messages

There may be times where you need to target one or more messages to a specific message component, for example suppose you have growl and messages on same page and you need to display some messages on growl and some on messages. Use for attribute to associate messages with specific components.

```
<p:messages for="somekey" />
<p:growl for="anotherkey" />
```

```
FacesContext context = FacesContext.getCurrentInstance();

context.addMessage("somekey", facesMessage1);
context.addMessage("somekey", facesMessage2);

context.addMessage("anotherkey", facesMessage3);
```

In sample above, messages will display first and second message and growl will only display the 3rd message.

Severity Levels

Using severity attribute, you can define which severities can be displayed by the component. For instance, you can configure messages to only display infos and warnings.

```
<p:messages severity="info,warn" />
```

Escaping

Messages escapes html content in messages, in case you need to display html, enable escape option.

```
<p:messages escape="true" />
```

Skinning Message

Full list of CSS selectors of message is as follows;

Style Class	Applies
ui-messages-{severity}	Container element of the message
ui-messages-{severity}-summary	Summary text
ui-messages-{severity}-detail	Detail text
ui-messages-{severity}-icon	Icon of the message.

{severity} can be 'info', 'error', 'warn' and error.

3.65 NotificationBar

NotificationBar displays a multipurpose fixed positioned panel for notification.

Info

Tag	**notificationBar**
Component Class	**org.primefaces.component.notificationbar.NotificationBar**
Component Type	**org.primefaces.component.NotificatonBar**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.NotificationBarRenderer**
Renderer Class	**org.primefaces.component.notificationbar.NotificationBarRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Style of the container element
styleClass	null	String	StyleClass of the container element
position	top	String	Position of the bar, "top" or "bottom".
effect	fade	String	Name of the effect, "fade", "slide" or "none".
effectSpeed	normal	String	Speed of the effect, "slow", "normal" or "fast".
autoDisplay	FALSE	Boolean	When true, panel is displayed on page load.

Getting started with NotificationBar

As notificationBar is a panel component, any content can be placed inside.

```
<p:notificationBar widgetVar="topBar">
      //Content
</p:notificationBar>
```

Showing and Hiding

To show and hide the content, notificationBar provides an easy to use client side api that can be accessed through the widgetVar. *show()* displays the bar and *hide()* hides it. *isVisible()* and *toggle()* are additional client side api methods.

```
<p:notificationBar widgetVar="topBar">
      //Content
</p:notificationBar>

<h:outputLink value="#" onclick="topBar.show()">Show</h:outputLink>
<h:outputLink value="#" onclick="topBar.show()">Show</h:outputLink>
```

Note that notificationBar has a default built-in close icon to hide the content.

Effects

Default effect to be used when displaying and hiding the bar is "fade", another possible effect is "slide".

```
<p:notificationBar widgetVar="topBar" effect="slide">
      //Content
</p:notificationBar>
```

If you'd like to turn off animation, set effect name to "none". In addition duration of the animation is controlled via effectSpeed attribute that can take "normal", "slow" or "fast" as it's value.

Position

Default position of bar is "top", other possibility is placing the bar at the bottom of the page. Note that bar positioning is fixed so even page is scrolled, bar will not scroll.

```
<p:notificationBar widgetVar="topBar" position="bottom">
      //Content
</p:notificationBar>
```

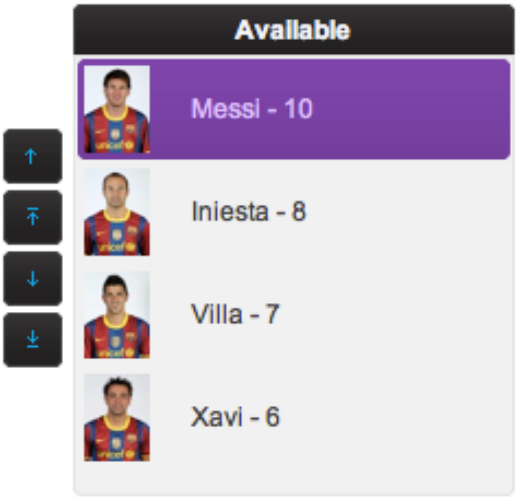
Skinning

style and styleClass attributes apply to the main container element. Additionally there are two pre-defined css selectors used to customize the look and feel.

Selector	Applies
.ui-notificationbar	Main container element
.ui-notificationbar-close	Close icon element

3.66 OrderList

OrderList is used to sort a collection featuring drag&drop based reordering, transition effects and pojo support.



Info

Tag	**orderList**
Component Class	**org.primefaces.component.orderlist.OrderList**
Component Type	**org.primefaces.component.OrderList**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.OrderListRenderer**
Renderer Class	**org.primefaces.component.orderlist.OrderListRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.

Name	Default	Type	Description
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
var	null	String	Name of the iterator.
itemLabel	null	String	Label of an item.
itemValue	null	String	Value of an item.
style	null	String	Inline style of container element.
styleClass	null	String	Style class of container element.
disabled	FALSE	Boolean	Disables the component.
effect	fade	String	Name of animation to display.
moveUpLabel	Move Up	String	Label of move up button.
moveTopLabel	Move Top	String	Label of move top button.
moveDownLabel	Move Down	String	Label of move down button.
moveBottomLabel	Move Bottom	String	Label of move bottom button.
controlsLocation	left	String	Location of the reorder buttons, valid values are "left", "right" and "none".

Getting started with OrderList

A list is required to use OrderList component.

```
public class OrderListBean {
      private List<String> cities;

      public OrderListBean() {
            cities = new ArrayList<String>();

            cities.add("Istanbul");
            cities.add("Ankara");
            cities.add("Izmir");
            cities.add("Antalya");
            cities.add("Bursa");
      }

      //getter&setter for cities
}
```

```
<p:orderList value="#{orderListBean.cities}" var="city"
                itemLabel="#{city}" itemValue="#{city}""/>
```

When the form is submitted, orderList will update the cities list according to the changes on client side.

Advanced OrderList

OrderList supports displaying custom content instead of simple labels by using columns. In addition, pojos are supported if a converter is defined.

```
public class OrderListBean {
      private List<Player> players;

      public OrderListBean() {
            players = new ArrayList<Player>();

            players.add(new Player("Messi", 10, "messi.jpg"));
            players.add(new Player("Iniesta", 8, "iniesta.jpg"));
            players.add(new Player("Villa", 7, "villa.jpg"));
            players.add(new Player("Xavi", 6, "xavi.jpg"));
      }

      //getter&setter for players
}
```

```
<p:orderList value="#{orderListBean.players}" var="player" itemValue="#{player}"
             converter="player">

             <p:column style="width:25%">
                    <p:graphicImage value="/images/barca/#{player.photo}" />
             </p:column>

             <p:column style="width:75%;">
                    #{player.name} - #{player.number}
             </p:column>
</p:orderList>
```

Header

A facet called "caption" is provided to display a header content for the orderlist.

Effects

An animation is executed during reordering, default effect is fade and following options are available for *effect* attribute;

- blind
- bounce
- clip
- drop
- explode
- fold
- highlight
- puff
- pulsate
- scale
- shake
- size
- slide

Skinning

OrderList resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-orderlist	Main container element.
.ui-orderlist-list	Container of items.
.ui-orderlist-item	Each item in the list.
.ui-orderlist-caption	Caption of the list.

3.67 OutputLabel

OutputPanel is a an extension to the standard outputLabel component.

Info

Tag	**outputLabel**
Component Class	**org.primefaces.component.outputlabel.OutputLabel**
Component Type	**org.primefaces.component.OutputLabel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.OutputLabelRenderer**
Renderer Class	**org.primefaces.component.outputlabel.OutputLabelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label to display.
accesskey	null	String	The accesskey attribute is a standard HTML attribute that sets the access key that transfers focus to this element when pressed.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
escape	TRUE	Boolean	Defines if value should be escaped or not.

Name	Default	Type	Description
for	null	String	Id of the associated input component.
tabindex	null	String	Position in tabbing order.
title	null	String	Advisory tooltip information.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
onblur	null	String	Client side callback to execute when component loses focus.
onclick	null	String	Client side callback to execute when component is clicked.
ondblclick	null	String	Client side callback to execute when component is double clicked.
onfocus	null	String	Client side callback to execute when component receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over component.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over component.
onkeyup	null	String	Client side callback to execute when a key is released over component.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over component.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from component.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto component.
onmouseup	null	String	Client side callback to execute when a pointer button is released over component.

Getting Started with OutputPanel

Usage is same as standard outputPanel, an input component is associated with for attribute.

```
<p:outputLabel for="input" value="Label" />

<p:inputText id="input" value="#{bean.text}" />
```

Auto Label

OutputLabel sets its value as the label of the target component to be displayed in validation errors so the target component does not need to define the label attribute again.

```
<h:outputLabel for="input" value="Field" />

<p:inputText id="input" value="#{bean.text}" label="Field"/>
```

can be rewritten as;

```
<p:outputLabel for="input" value="Field" />

<p:inputText id="input" value="#{bean.text}" />
```

Support for Advanced Components

Some PrimeFaces input components like spinner, autocomplete does not render just basic inputs so standard outputLabel component cannot apply focus to these, however PrimeFaces outputPanel can.

```
<h:outputLabel for="input" value="Can't apply focus" />
<p:outputLabel for="input" value="Can apply focus" />

<p:spinner id="input" value="#{bean.text}" />
```

Validations

When the target input is required, outputLabel displays * symbol next to the value. In case any validation fails on target input, label will also be displayed with theme aware error styles.

Skinning

Label renders a label element that *style* and *styleClass* attributes apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-outputlabel	Label element
.ui-state-error	Label element when input is invalid
.ui-outputlabel-rfi	Required field indicator.

3.68 OutputPanel

OutputPanel is a panel component with the ability to auto update.

Info

Tag	**outputPanel**
Component Class	**org.primefaces.component.outputpanel.OutputPanel**
Component Type	**org.primefaces.component.OutputPanel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.OutputPanelRenderer**
Renderer Class	**org.primefaces.component.output.OutputPanelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Style of the html container element
styleClass	null	String	StyleClass of the html container element
layout	inline	String	Layout of the panel, valid values are *inline*(span) or *block*(div).
autoUpdate	FALSE	Boolean	Enables auto update mode if set true.

AjaxRendered

Due to the nature of ajax, it is much simpler to update an existing element on page rather than inserting a new element to the dom. When a JSF component is not rendered, no markup is rendered so for components with conditional rendering regular PPR mechanism may not work since the markup to update on page does not exist. OutputPanel is useful in this case.

Suppose the rendered condition on bean is false when page if loaded initially and search method on bean sets the condition to be true meaning datatable will be rendered after a page submit. The problem is although partial output is generated, the markup on page cannot be updated since it doesn't exist.

```
<p:dataTable id="tbl" rendered="#{bean.condition}" ...>
    //columns
</p:dataTable>

<p:commandButton update="tbl" actionListener="#{bean.search}" />
```

Solution is to use the outputPanel as a placeHolder.

```
<p:outputPanel id="out">
    <p:dataTable id="tbl" rendered="#{bean.condition}" ...>
        //columns
    </p:dataTable>
</p:outputPanel>

<p:commandButton update="out" actionListener="#{bean.list}" />
```

Note that you won't need an outputPanel if commandButton has no update attribute specified, in this case parent form will be updated partially implicitly making an outputPanel use obselete.

Layout

OutputPanel has two layout modes;

- inline (default): Renders a span
- block: Renders a div

AutoUpdate

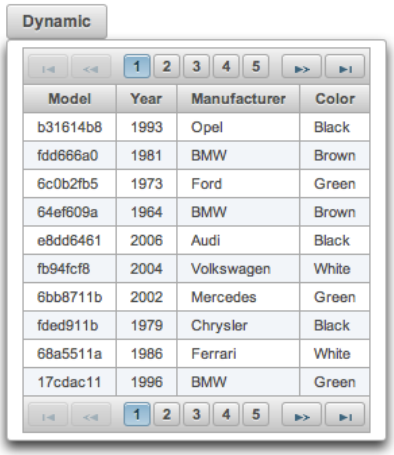
When auto update is enabled, outputPanel component is updated with each ajax request automatically.

Skinning OutputPanel

style and *styleClass* attributes are used to skin the outputPanel.

3.69 OverlayPanel

OverlayPanel is a generic panel component that can be displayed on top of other content.



Info

Tag	**overlayPanel**
Component Class	**org.primefaces.component.overlaypanel.OverlayPanelRenderer**
Component Type	**org.primefaces.component.OverlayPanel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.OverlayPanelRenderer**
Renderer Class	**org.primefaces.component.overlaypanel.OverlayPanelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
style	null	String	Inline style of the panel.
styleClass	null	String	Style class of the panel.
for	null	String	Identifier of the target component to attach the panel.

Name	Default	Type	Description
showEvent	mousedown	String	Event on target to show the panel.
hideEvent	mousedown	String	Event on target to hide the panel.
showEffect	null	String	Animation to display when showing the panel.
hideEffect	null	String	Animation to display when hiding the panel.
appendToBody	FALSE	Boolean	When true, panel is appended to document body.
onShow	null	String	Client side callback to execute when panel is shown.
onHide	null	String	Client side callback to execute when panel is hidden.
my	left top	String	Position of the panel relative to the target.
at	left bottom	String	Position of the target relative to the panel.
dynamic	FALSE	Boolean	Defines content loading mode.

Getting started with OverlayPanel

OverlayPanel needs a component as a target in addition to the content to display. Example below demonstrates an overlayPanel attached to a button to show a chart in a popup.

```
 <p:commandButton id="chartBtn" value="Basic" type="button" />

<p:overlayPanel for="chartBtn">
      <p:pieChart value="#{chartBean.pieModel}" legendPosition="w"
            title="Sample Pie Chart" style="width:400px;height:300px" />
</p:overlayPanel>
```

Events

Default event on target to show and hide the panel is mousedown. These are customized using *showEvent* and *hideEvent* options.

```
 <p:commandButton id="chartBtn" value="Basic" type="button" />

<p:overlayPanel showEvent="mouseover" hideEvent="mousedown">
      //content
</p:overlayPanel>
```

Effects

blind, bounce, clip, drop, explode, fold, highlight, puff, pulsate, scale, shake, size, slide are available values for *showEffect* and *hideEffect* options if you'd like display animations.

Positioning

By default, left top corner of panel is aligned to left bottom corner of the target if there is enough space in window viewport, if not the position is flipped on the fly to find the best location to display. In order to customize the position use *my* and *at* options that takes combinations of left, right, bottom and top e.g. "right bottom".

Dynamic Mode

Dynamic mode enables lazy loading of the content, in this mode content of the panel is not rendered on page load and loaded just before panel is shown. Also content is cached so consecutive displays do not load the content again. This feature is useful to reduce the page size and reduce page load time.

Skinning Panel

Panel resides in a main container which *style* and *styleClass* attributes apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-overlaypanel	Main container element of panel

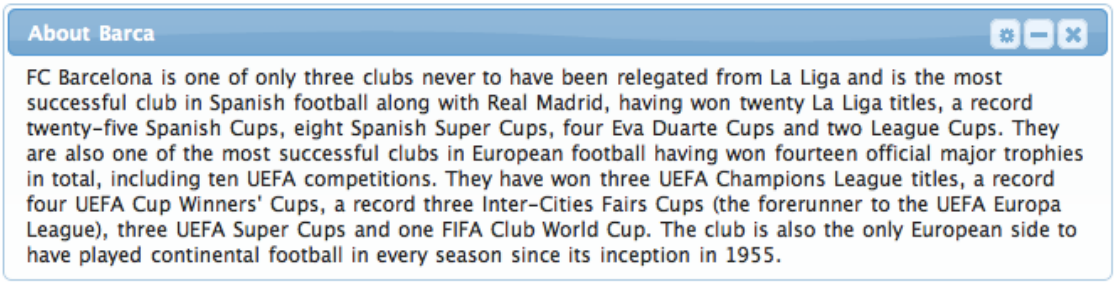
As skinning style classes are global, see the main Skinning section for more information.

Tips

- Enable appendToBody when overlayPanel is in other panel components like layout, dialog ...

3.70 Panel

Panel is a grouping component with content toggle, close and menu integration.


About Barca
FC Barcelona is one of only three clubs never to have been relegated from La Liga and is the most successful club in Spanish football along with Real Madrid, having won twenty La Liga titles, a record twenty-five Spanish Cups, eight Spanish Super Cups, four Eva Duarte Cups and two League Cups. They are also one of the most successful clubs in European football having won fourteen official major trophies in total, including ten UEFA competitions. They have won three UEFA Champions League titles, a record four UEFA Cup Winners' Cups, a record three Inter-Cities Fairs Cups (the forerunner to the UEFA Europa League), three UEFA Super Cups and one FIFA Club World Cup. The club is also the only European side to have played continental football in every season since its inception in 1955.

Info

Tag	**panel**
Component Class	**org.primefaces.component.panel.Panel**
Component Type	**org.primefaces.component.Panel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PanelRenderer**
Renderer Class	**org.primefaces.component.panel.PanelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
header	null	String	Header text
footer	null	String	Footer text
toggleable	FALSE	Boolean	Makes panel toggleable.
toggleSpeed	1000	Integer	Speed of toggling in milliseconds
collapsed	FALSE	Boolean	Renders a toggleable panel as collapsed.
style	null	String	Style of the panel
styleClass	null	String	Style class of the panel

Name	Default	Type	Description
closable	FALSE	Boolean	Make panel closable.
closeSpeed	1000	Integer	Speed of closing effect in milliseconds
visible	TRUE	Boolean	Renders panel as visible.
closeTitle	null	String	Tooltip for the close button.
toggleTitle	null	String	Tooltip for the toggle button.
menuTitle	null	String	Tooltip for the menu button.
widgetVar	null	String	Name of the client side widget

Getting started with Panel

Panel encapsulates other components.

```
<p:panel>
      //Child components here...
</p:panel>
```

Header and Footer

Header and Footer texts can be provided by *header* and *footer* attributes or the corresponding facets. When same attribute and facet name are used, facet will be used.

```
<p:panel header="Header Text">
      <f:facet name="footer">
            <h:outputText value="Footer Text" />
      </f:facet>

      //Child components here...
</p:panel>
```

Ajax Behavior Events

Panel provides custom ajax behavior events for toggling and closing features.

Event	Listener Parameter	Fired
toggle	org.primefaces.event.ToggleEvent	When panel is expanded or collapsed.
close	org.primefaces.event.CloseEvent	When panel is closed.

Popup Menu

Panel has built-in support to display a fully customizable popup menu, an icon to display the menu is placed at top-right corner. This feature is enabled by defining a menu component and defining it as the options facet.

```
<p:panel closable="true">
      //Child components here...

      <f:facet name="options">
            <p:menu>
                  //Menuitems
            </p:menu>
      </f:facet>
</p:panel>
```

Skinning Panel

Panel resides in a main container which *style* and *styleClass* attributes apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-panel	Main container element of panel
.ui-panel-titlebar	Header container
.ui-panel-title	Header text
.ui-panel-titlebar-icon	Option icon in header
.ui-panel-content	Panel content
.ui-panel-footer	Panel footer

As skinning style classes are global, see the main Skinning section for more information.

3.71 PanelGrid

PanelGrid is an extension to the standard panelGrid component with additional features such as theming and colspan-rowspan.

1995-96 NBA Playoffs						
Conf. Semifinals		Conf. Finals		NBA Finals		Champion
Seattle	4	Seattle	4	Seattle	2	Chicago
Houston	0					
Utah	4	Utah	3			
San Antonio	2					
Chicago	4	Chicago	4	Chicago	4	
New York	1					
Atlanta	1	Orlando	0			
Orlando	4					
Finals MVP				Michael Jordan (Chicago)		
Season MVP						
Top Scorer						

Info

Tag	**panelGrid**
Component Class	**org.primefaces.component.panelgrid.PanelGridRenderer**
Component Type	**org.primefaces.component.PanelGrid**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PanelGridRenderer**
Renderer Class	**org.primefaces.component.panelgrid.PanelGridRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Name	Default	Type	Description
columns	0	Integer	Number of columns in grid.
style	null	String	Inline style of the panel.
styleClass	null	String	Style class of the panel.
columnClasses	null	String	Comma separated list of column style classes.

Getting started with PanelGrid

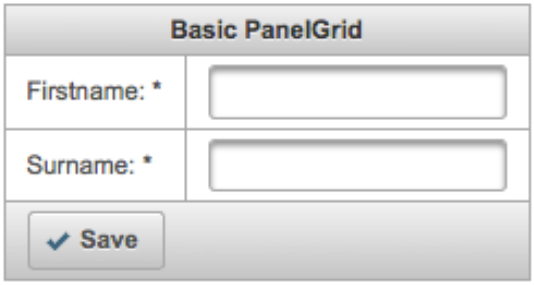
Basic usage of panelGrid is same as the standard one.

```
<p:panelGrid columns="2">
      <h:outputLabel for="firstname" value="Firstname:" />
      <p:inputText id="firstname" value="#{bean.firstname}" label="Firstname" />

      <h:outputLabel for="surname" value="Surname:" />
      <p:inputText id="surname" value="#{bean.surname}" label="Surname"/>
</p:panelGrid>
```

Header and Footer

PanelGrid provides facets for header and footer content.

```
<p:panelGrid columns="2">
      <f:facet name="header">
            Basic PanelGrid
      </f:facet>

      <h:outputLabel for="firstname" value="Firstname: *" />
      <p:inputText id="firstname" value="#{bean.firstname}" label="Firstname" />

      <h:outputLabel for="surname" value="Surname: *" />
      <p:inputText id="surname" value="#{bean.surname}" label="Surname"/>

      <f:facet name="footer">
            <p:commandButton type="button" value="Save" icon="ui-icon-check" />
      </f:facet>
</p:panelGrid>
```



Rowspan and Colspan

PanelGrid supports rowspan and colspan options as well, in this case row and column markup should be defined manually.

```
<p:panelGrid>
      <p:row>
            <p:column rowspan="3">AAA</p:column>
            <p:column colspan="4">BBB</p:column>
      </p:row>

      <p:row>
            <p:column colspan="2">CCC</p:column>
            <p:column colspan="2">DDD</p:column>
      </p:row>

      <p:row>
            <p:column>EEE</p:column>
            <p:column>FFF</p:column>
            <p:column>GGG</p:column>
            <p:column>HHH</p:column>
      </p:row>
</p:panelGrid>
```

Skinning PanelGrid

PanelGrid resides in a main container which *style* and *styleClass* attributes apply.

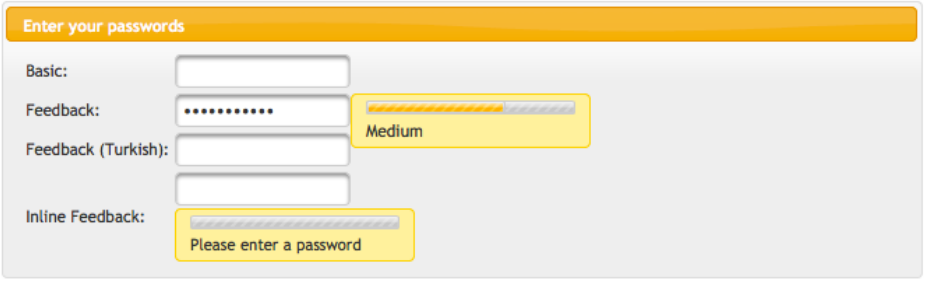
Following is the list of structural style classes;

Style Class	Applies
.ui-panelgrid	Main container element of panelGrid.
.ui-panelgrid-header	Header.
.ui-panelgrid-footer	Footer.

As skinning style classes are global, see the main Skinning section for more information.

3.72 Password

Password component is an extended version of standard inputSecret component with theme integration and strength indicator.



Info

Tag	**password**
Component Class	**org.primefaces.component.password.Password**
Component Type	**org.primefaces.component.Password**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PasswordRenderer**
Renderer Class	**org.primefaces.component.password.PasswordRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	boolean	Marks component as required

Name	Default	Type	Description
validator	null	MethodExpr	A method expression that refers to a method validationg the input.
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
feedback	FALSE	Boolean	Enables strength indicator.
inline	FALSE	boolean	Displays feedback inline rather than using a popup.
promptLabel	Please enter a password	String	Label of prompt.
level	1	Integer	Level of security.
weakLabel	Weak	String	Label of weak password.
goodLabel	Good	String	Label of good password.
strongLabel	Strong	String	Label of strong password.
redisplay	FALSE	Boolean	Whether or not to display previous value.
match	null	String	Id of another password component to match value against.
widgetVar	null	String	Name of the client side widget.
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.

Name	Default	Type	Description
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Getting Started with Password

Password is an input component and used just like a standard input text. When *feedback* option is enabled a password strength indicator is displayed.

```
<p:password value="#{bean.password}" feedback="true|false" />
```

```
public class Bean {

      private String password;

      public String getPassword() { return password; }
      public void setPassword(String password) { this.password = password;}
}
```

I18N

Although all labels are in English by default, you can provide custom labels as well. Following password gives feedback in Turkish.

```
<p:password value="#{bean.password}" promptLabel="Lütfen şifre giriniz"
weakLabel="Zayıf" goodLabel="Orta seviye" strongLabel="Güçlü" feedback= "true"/>
```

Inline Strength Indicator

By default strength indicator is shown in an overlay, if you prefer an inline indicator just enable inline mode.

```
<p:password value="#{mybean.password}" inline="true" feedback= "true"/>
```

Confirmation

Password confirmation is a common case and password provides an easy way to implement. The other password component's id should be used to define the *match* option.

```
<p:password id="pwd1" value="#{passwordBean.password6}" feedback="false"
      match="pwd2" label="Password 1" required="true"/>

<p:password id="pwd2" value="#{passwordBean.password6}" feedback="false"
      label="Password 2" required="true"/>
```

Skinning

Structural selectors for password is as follows;

Name	Applies
.ui-password	Input element.
.ui-password-panel	Overlay panel of strength indicator.
.ui-password-meter	Strength meter.
.ui-password-info	Strength label.

As skinning style classes are global, see the main Skinning section for more information.

3.73 PhotoCam

PhotoCam is used to take photos with webcam and send them to the JSF backend model.

Info

Tag	**photoCam**
Component Class	**org.primefaces.component.photocam.PhotoCam**
Component Type	**org.primefaces.component.PhotoCam**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PhotoCamRenderer**
Renderer Class	**org.primefaces.component.photocam.PhotoCamRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	boolean	Marks component as required
validator	null	MethodBind ing	A method binding expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.

Name	Default	Type	Description
widgetVar	null	String	Name of the client side widget.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
process	null	String	Identifiers of components to process during capture.
update	null	String	Identifiers of components to update during capture.
listener	null	MethodExpr	Method expression to listen to capture events.

Getting started with PhotoCam

Capture is triggered via client side api's *capture* method. Also a method expression is necessary to invoke when an image is captured. Sample below captures an image and saves it to a directory.

```
<h:form>
   <p:photoCam widgetVar="pc" listener="#{photoCamBean.oncapture}"update="photos"/>

   <p:commandButton type="button" value="Capture" onclick="pc.capture()"/>
</h:form>
```

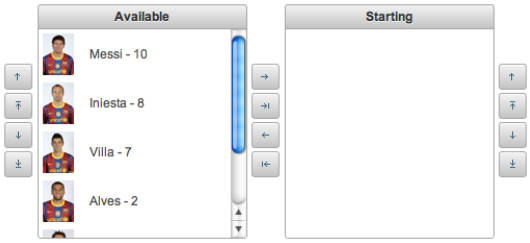
```
public class PhotoCamBean {

   public void oncapture(CaptureEvent captureEvent) {
       byte[] data = captureEvent.getData();

            ServletContext servletContext = (ServletContext)
FacesContext.getCurrentInstance().getExternalContext().getContext();
            String newFileName = servletContext.getRealPath("") + File.separator +
"photocam" + File.separator + "captured.png";

            FileImageOutputStream imageOutput;
            try {
                  imageOutput = new FileImageOutputStream(new File(newFileName));
                  imageOutput.write(data, 0, data.length);
                  imageOutput.close();
            }
       catch(Exception e) {
                  throw new FacesException("Error in writing captured image.");
            }
   }
}
```

Notes

- PhotoCam is a flash, canvas and javascript solution.
- It is not supported in IE at the moment and this will be worked on in future versions.

3.74 PickList

PickList is used for transferring data between two different collections.



Info

Tag	**pickList**
Component Class	**org.primefaces.component.picklist.Panel**
Component Type	**org.primefaces.component.PickList**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PickListRenderer**
Renderer Class	**org.primefaces.component.picklist.PickListRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required

Name	Default	Type	Description
validator	null	Method Expr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	Method Expr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
var	null	String	Name of the iterator.
itemLabel	null	String	Label of an item.
itemValue	null	Object	Value of an item.
style	null	String	Style of the main container.
styleClass	null	String	Style class of the main container.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
effect	null	String	Name of the animation to display.
effectSpeed	null	String	Speed of the animation.
addLabel	Add	String	Title of add button.
addAllLabel	Add All	String	Title of add all button.
removeLabel	Remove	String	Title of remove button.
removeAllLabel	Remove All	String	Title of remove all button.
moveUpLabel	Move Up	String	Title of move up button.
moveTopLabel	Move Top	String	Title of move top button.
moveDownLabel	Move Down	String	Title of move down button.
moveButtomLabel	Move Buttom	String	Title of move bottom button.
showSourceControls	FALSE	String	Specifies visibility of reorder buttons of source list.
showTargetControls	FALSE	String	Specifies visibility of reorder buttons of target list.
onTransfer	null	String	Client side callback to execute when an item is transferred from one list to another.
label	null	String	A localized user presentable name.
itemDisabled	FALSE	Boolean	Specified if an item can be picked or not.

Getting started with PickList

You need to create custom model called *org.primefaces.model.DualListModel* to use PickList. As the name suggests it consists of two lists, one is the source list and the other is the target. As the first example we'll create a DualListModel that contains basic Strings.

```
public class PickListBean {

      private DualListModel<String> cities;

      public PickListBean() {
            List<String> source = new ArrayList<String>();
            List<String> target = new ArrayList<String>();

            citiesSource.add("Istanbul");
            citiesSource.add("Ankara");
            citiesSource.add("Izmir");
            citiesSource.add("Antalya");
            citiesSource.add("Bursa");

            //more cities

            cities = new DualListModel<String>(citiesSource, citiesTarget);
      }

      public DualListModel<String> getCities() {
            return cities;
      }

      public void setCities(DualListModel<String> cities) {
            this.cities = cities;
      }
}
```

And bind the cities dual list to the picklist;

```
<p:pickList value="#{pickListBean.cities}" var="city"
            itemLabel="#{city}" itemValue="#{city}">
```

When the enclosed form is submitted, the dual list reference is populated with the new values and you can access these values with DualListModel.getSource() and DualListModel.getTarget() api.

POJOs

Most of the time you would deal with complex pojos rather than simple types like String.
This use case is no different except the addition of a converter.

Following pickList displays a list of players(name, age ...).

```
public class PickListBean {

      private DualListModel<Player> players;

      public PickListBean() {
            //Players
            List<Player> source = new ArrayList<Player>();
            List<Player> target = new ArrayList<Player>();

            source.add(new Player("Messi", 10));
            //more players

            players = new DualListModel<Player>(source, target);
      }

      public DualListModel<Player> getPlayers() {
            return players;
      }
      public void setPlayers(DualListModel<Player> players) {
            this.players = players;
      }
}
```

```
<p:pickList value="#{pickListBean.players}" var="player"
      itemLabel="#{player.name}" itemValue="#{player}" converter="player">
```

PlayerConverter in this case should implement *javax.faces.convert.Converter* contract and implement getAsString, getAsObject methods. Note that a converter is always necessary for primitive types like long, integer, boolean as well.

Custom content instead of simple strings can be displayed by using columns.

```
<p:pickList value="#{pickListBean.players}"
                    var="player" iconOnly="true" effect="bounce"
                    itemValue="#{player}" converter="player"
                    showSourceControls="true" showTargetControls="true">
        <p:column style="width:25%">
            <p:graphicImage value="/images/barca/#{player.photo}"/>
        </p:column>

         <p:column style="width:75%";>
            #{player.name} - #{player.number}
        </p:column>
</p:pickList>
```

Reordering

PickList support reordering of source and target lists, these are enabled by *showSourceControls* and *showTargetControls* options.

Effects

An animation is displayed when transferring when item to another or reordering a list, default effect is fade and following options are available to be applied using *effect* attribute;

- blind
- bounce
- clip
- drop
- explode
- fold
- highlight
- puff
- pulsate
- scale
- shake
- size
- slide

effectSpeed attribute is used to customize the animation speed, valid values are *slow*, *normal* and *fast*.

onTransfer

If you'd like to execute custom javascript when an item is transferred bind your javascript function to *onTransfer* attribute.

```
<p:pickList value="#{pickListBean.cities}" var="city"
            itemLabel="#{city}" itemValue="#{city}" onTransfer="handleTransfer(e)">
```

```
<script type="text/javascript">
     function handleTransfer(e) {
           //item = e.item
           //fromList = e.from
           //toList = e.toList
           //type = e.type (type of transfer; command, dblclick or dragdrop)
     }
</script>
```

Captions

Caption texts for lists are defined with facets named *sourceCaption* and *targetCaption*;

```
<p:pickList value="#{pickListBean.cities}" var="city"
            itemLabel="#{city}" itemValue="#{city}" onTransfer="handleTransfer(e)">
     <f:facet name="sourceCaption">Available</facet>
     <f:facet name="targetCaption">Selected</facet>
</p:picklList>
```

Skinning

PickList resides in a main container which *style* and *styleClass* attributes apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-picklist	Main container element(table) of picklist
.ui-picklist-list	Lists of a picklist
.ui-picklist-list-source	Source list
.ui-picklist-list-target	Target list
.ui-picklist-source-controls	Container element of source list reordering controls
.ui-picklist-target-controls	Container element of target list reordering controls
.ui-picklist-button	Buttons of a picklist
.ui-picklist-button-move-up	Move up button
.ui-picklist-button-move-top	Move top button
.ui-picklist-button-move-down	Move down button
.ui-picklist-button-move-bottom	Move bottom button
.ui-picklist-button-add	Add button
.ui-picklist-button-add-all	Add all button
.ui-picklist-button-remove-all	Remove all button
.ui-picklist-button-add	Add button

As skinning style classes are global, see the main Skinning section for more information.

3.75 Poll

Poll is an ajax component that has the ability to send periodical ajax requests.

Info

Tag	**poll**
Component Class	**org.primefaces.component.poll.Poll**
Component Type	**org.primefaces.component.Poll**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PollRenderer**
Renderer Class	**org.primefaces.component.poll.PollRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Name of the client side widget.
interval	2	Integer	Interval in seconds to do periodic ajax requests.
update	null	String	Component(s) to be updated with ajax.
listener	null	MethodExpr	A method expression to invoke by polling.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component id(s) to process partially instead of whole view.
onstart	null	String	Javascript handler to execute before ajax request is begins.
oncomplete	null	String	Javascript handler to execute when ajax request is completed.
onsuccess	null	String	Javascript handler to execute when ajax request succeeds.
onerror	null	String	Javascript handler to execute when ajax request fails.

Name	Default	Type	Description
global	TRUE	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
autoStart	TRUE	Boolean	In autoStart mode, polling starts automatically on page load, to start polling on demand set to false.
stop	FALSE	Boolean	Stops polling when true.

Getting started with Poll

Poll below invokes increment method on CounterBean every 2 seconds and *txt_count* is updated with the new value of the count variable. Note that poll must be nested inside a form.

```
<h:outputText id="txt_count" value="#{counterBean.count}" />

<p:poll listener="#{counterBean.increment}" update="txt_count" />
```

```
public class CounterBean {

      private int count;

      public void increment() {
            count++;
      }

      public int getCount() {
            return this.count;
      }

      public void setCount(int count) {
            this.count = count;
      }
}
```

Tuning timing

By default the periodic interval is 2 seconds, this is changed with the interval attribute. Following poll works every 5 seconds.

```
<h:outputText id="txt_count" value="#{counterBean.count}" />

<p:poll listener="#{counterBean.increment}" update="txt_count" interval="5" />
```

Start and Stop

Poll can be started and stopped using client side api;

```
<h:form>

      <h:outputText id="txt_count" value="#{counterBean.count}" />

      <p:poll interval="5" actionListener="#{counterBean.increment}"
                  update="txt_count" widgetVar="myPoll" autoStart="false" />

      <a href="#" onclick="myPoll.start();">Start</a>
      <a href="#" onclick="myPoll.stop();">Stop</a>

</h:form>
```

Or bind a boolean variable to the *stop* attribute and set it to false at any arbitrary time.

3.76 Printer

Printer allows sending a specific JSF component to the printer, not the whole page.

Info

Tag	**printer**
Behavior Class	**org.primefaces.component.behavior.Printer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
target	null	String	Id of the component to print.

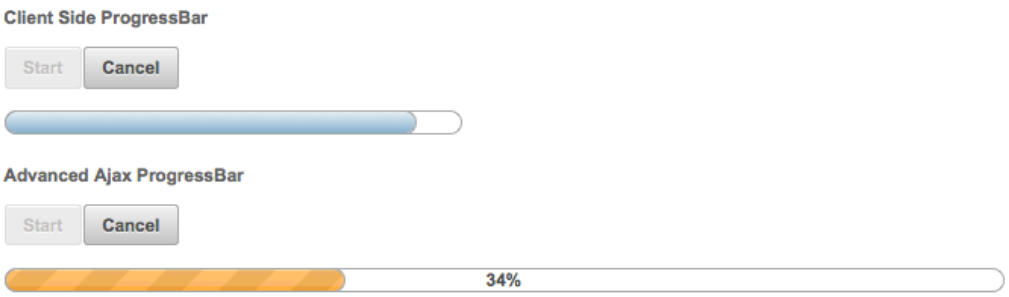
Getting started with the Printer

Printer is attached to any command component like a button or a link. Examples below demonstrates how to print a simple output text or a particular image on page;

```
 <h:commandButton id="btn" value="Print">
      <p:printer target="output" />
</h:commandButton>
<h:outputText id="output" value="PrimeFaces Rocks!" />

<h:outputLink id="lnk" value="#">
      <p:printer target="image" />
      <h:outputText value="Print Image" />
</h:outputLink>
<p:graphicImage id="image" value="/images/nature1.jpg" />
```

3.77 ProgressBar

ProgressBar is a process status indicator that can either work purely on client side or interact with server side using ajax.



Info

Tag	**propressBar**
Component Class	**org.primefaces.component.progressbar.ProgressBar**
Component Type	**org.primefaces.component.Progressbar**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ProgressBarRenderer**
Renderer Class	**org.primefaces.component.progressbar.ProgressBarRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget
value	0	Integer	Value of the progress bar
disabled	FALSE	Boolean	Disables or enables the progressbar
ajax	FALSE	Boolean	Specifies the mode of progressBar, in ajax mode progress value is retrieved from a backing bean.
interval	3000	Integer	Interval in seconds to do periodic requests in ajax mode.

Name	Default	Type	Description
style	null	String	Inline style of the main container element.
styleClass	null	String	Style class of the main container element.
labelTemplate	{value}	String	Template of the progress label.

Getting started with the ProgressBar

ProgressBar has two modes, "client"(default) or "ajax". Following is a pure client side progressBar.

```
 <p:progressBar widgetVar="pb" />

 <p:commandButton value="Start" type="button" onclick="start()" />
 <p:commandButton value="Cancel" type="button" onclick="cancel()" />

<script type="text/javascript">
      function start() {
            this.progressInterval = setInterval(function(){
                  pb.setValue(pbClient.getValue() + 10);
            }, 2000);
      }

      function cancel() {
            clearInterval(this.progressInterval);
            pb.setValue(0);
      }
</script>
```

Ajax Progress

Ajax mode is enabled by setting ajax attribute to true, in this case the value defined on a managed bean is retrieved periodically and used to update the progress.

```
<p:progressBar ajax="true" value="#{progressBean.progress}" />
```

```
public class ProgressBean {

      private int progress;

      //getter and setter
}
```

Interval

ProgressBar is based on polling and 3000 milliseconds is the default interval for ajax progress bar meaning every 3 seconds progress value will be recalculated. In order to set a different value, use the interval attribute.

```
<p:progressBar interval="5000" />
```

Ajax Behavior Events

ProgressBar provides *complete* as the default and only ajax behavior event that is fired when the progress is completed. Example below demonstrates how to use this event.

```
public class ProgressBean {

      private int progress;

      public void handleComplete() {
            //Add a faces message
      }

      //getter-setter
}
```

```
<p:progressBar value="#{progressBean.progress}" ajax="true">
      <p:ajax event="complete" listener="#{progressBean.handleComplete}"
                  update="messages" />
</p:progressBar>

<p:growl id="messages" />
```

Client Side API

Widget: *PrimeFaces.widget.ProgressBar*

Method	Params	Return Type	Description
getValue()	-	Number	Returns current value
setValue(value)	value: Value to display	void	Sets current value
start()	-	void	Starts ajax progress bar
cancel()	-	void	Stops ajax progress bar

Skinning

ProgressBar resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-progressbar	Main container.
.ui-progressbar-value	Value of the progressbar
.ui-progressbar-label	Progress label.

As skinning style classes are global, see the main Skinning section for more information.

3.78 Push

Push component is an agent that creates a channel between the server and the client.

Info

Tag	**push**
Component Class	**org.primefaces.component.push.Push**
Component Type	**org.primefaces.component.Push**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.PushRenderer**
Renderer Class	**org.primefaces.component.push.PushRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
channel	null	Object	Unique channel name of the connection between subscriber and the server.
onmessage	null	String	Client side callback to execute when data is pushed.
onclose	null	String	Client side callback to execute when connection is closed.
autoConnect	TRUE	Boolean	Connects to channel on page load when enabled.

Getting Started with the Push

See chapter 6, "PrimeFaces Push" for detailed information.

Client Side API of PrimeFaces.widget.PrimeWebSocket

Method	Description
void send(data)	Sends data in json format to push server.
void connect()	Connects to the channel.
void close()	Disconnects from channel.

3.79 RadioButton

RadioButton is a helper component of SelectOneRadio to implement custom layouts.

Info

Tag	**radioButton**
Component Class	**org.primefaces.component.radiobutton.RadioButton**
Component Type	**org.primefaces.component.RadioButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RadioButtonRenderer**
Renderer Class	**org.primefaces.component.radiobutton.RadioButtonRenderer**

Attributes

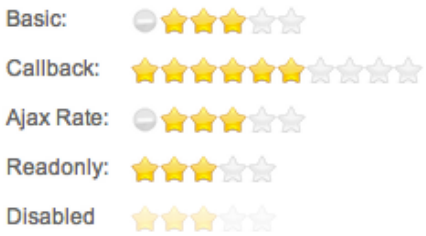
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
disabled	FALSE	Boolean	Disabled the component.
itemIndex	null	Integer	Index of the selectItem of selectOneRadio.
onchange	null	String	Client side callback to execute on state change.
for	null	String	Id of the selectOneRadio to attach to.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with RadioButton

See custom layout part in SelectOneRadio section for more information.

3.80 Rating

Rating component features a star based rating system.



Info

Tag	**rating**
Component Class	**org.primefaces.component.rating.Rating**
Component Type	**org.primefaces.component.Rating**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RatingRenderer**
Renderer Class	**org.primefaces.component.rating.RatingRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id

Name	Default	Type	Description
immediate	FALSE	Boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method binding expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
stars	5	Integer	Number of stars to display
disabled	FALSE	Boolean	Disables user interaction
readonly	FALSE	Boolean	Disables user interaction without disabled visuals.
onRate	null	String	Client side callback to execute when rate happens.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
cancel	TRUE	Boolean	When enabled, displays a cancel icon to reset.

Getting Started with Rating

Rating is an input component that takes a double variable as it's value.

```
public class RatingBean {

      private Integer rating;

      //getter-setter
}
```

```
<p:rating value="#{ratingBean.rating}" />
```

Number of Stars

Default number of stars is 5, if you need less or more stars use the stars attribute. Following rating consists of 10 stars.

```
<p:rating value="#{ratingBean.rating}" stars="10"/>
```

Display Value Only

In cases where you only want to use the rating component to display the rating value and disallow user interaction, set *readonly* to true. Using *disabled* attribute does the same but adds disabled visual styles.

Ajax Behavior Events

Rating provides *rate* and *cancel* as ajax behavior events. A defined listener for rate event will be executed by passing an *org.primefaces.event.RateEvent* as a parameter and cancel event will be invoked with no parameter.

```
<p:rating value="#{ratingBean.rating}">
      <p:ajax event="rate" listener="#{ratingBean.handleRate}" update="msgs" />
      <p:ajax event="cancel" listener="#{ratingBean.handleCancel}" update="msgs" />
</p:rating>
 <p:messages id="msgs" />
```

```
public class RatingBean {

      private Integer rating;

      public void handleRate(RateEvent rateEvent) {
            Integer rate = (Integer) rateEvent.getRating();
            //Add facesmessage
      }

      public void handleCancel() {
            //Add facesmessage
      }

      //getter-setter
}
```

Client Side Callbacks

onRate is called when a star is selected with *value* as the only parameter.

```
<p:rating value="#{ratingBean.rating}" onRate="alert('You rated: ' + value)" />
```

Client Side API

Widget: *PrimeFaces.widget.Rating*

Method	Params	Return Type	Description
getValue()	-	Number	Returns the current value
setValue(value)	value: Value to set	void	Updates rating value with provided one.
disable()	-	void	Disables component.
enable()	-	void	Enables component.
reset()	-	void	Clears the rating.

Skinning

ProgressBar resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-rating	Main container element.
.ui-rating-cancel	Cancel icon
.ui-rating	Default star
.ui-rating-on	Active star

3.81 RemoteCommand

RemoteCommand provides a way to execute JSF backing bean methods directly from javascript.

Info

Tag	**remoteCommand**
Component Class	**org.primefaces.component.remotecommand.RemoteCommand**
Component Type	**org.primefaces.component.RemoteCommand**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RemoteCommandRenderer**
Renderer Class	**org.primefaces.component.remotecommand.RemoteCommandRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolea	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
action	null	MethodExpr	A method expression that'd be processed in the partial request caused by uiajax.
actionListener	null	MethodExpr	An actionlistener that'd be processed in the partial request caused by uiajax.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
name	null	String	Name of the command
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component(s) to process partially instead of whole view.
update	null	String	Component(s) to update with ajax.
onstart	null	String	Javascript handler to execute before ajax request is begins.
oncomplete	null	String	Javascript handler to execute when ajax request is completed.
onsuccess	null	String	Javascript handler to execute when ajax request succeeds.
onerror	null	String	Javascript handler to execute when ajax request fails.

Name	Default	Type	Description
global	TRUE	Boolean	Global ajax requests are listened by ajaxStatus component, setting global to false will not trigger ajaxStatus.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
autoRun	FALSE	Boolean	When enabled command is executed on page load.

Getting started with RemoteCommand

RemoteCommand is used by invoking the command from your javascript code.

```
<p:remoteCommand name="increment" actionListener="#{counter.increment}"
      out="count" />

<h:outputText id="count" value="#{counter.count}" />
```

```
<script type="text/javascript">
      function customfunction() {
            //your custom code

            increment();        //makes a remote call
      }
</script>
```

That's it whenever you execute your custom javascript function(eg customfunction()), a remote call will be made, actionListener is processed and output text is updated. Note that remoteCommand must be nested inside a form.

Passing Parameters

Remote command can send dynamic parameters in the following way;

```
increment([{name:'x', value:10}, {name:'y', value:20}]);
```

Run on Load

If you'd like to run the command on page load, set autoRun to true.

3.82 Resizable

Resizable component is used to make another JSF component resizable.

Info

Tag	**resizable**
Component Class	**org.primefaces.component.resizable.Resizable**
Component Type	**org.primefaces.component.Resizable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ResizableRenderer**
Renderer Class	**org.primefaces.component.resizable.ResizableRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
for	null	String	Identifier of the target component to make resizable.
aspectRatio	FALSE	Boolean	Defines if aspectRatio should be kept or not.
proxy	FALSE	Boolean	Displays proxy element instead of actual element.
handles	null	String	Specifies the resize handles.
ghost	FALSE	Boolean	In ghost mode, resize helper is displayed as the original element with less opacity.
animate	FALSE	Boolean	Enables animation.
effect	swing	String	Effect to use in animation.
effectDuration	normal	String	Effect duration of animation.
maxWidth	null	Integer	Maximum width boundary in pixels.
maxHeight	null	Integer	Maximum height boundary in pixels.
minWidth	10	Integer	Minimum width boundary in pixels.
minHeight	10	Integer	Maximum height boundary in pixels.

Name	Default	Type	Description
containment	FALSE	Boolean	Sets resizable boundaries as the parents size.
grid	1	Integer	Snaps resizing to grid structure.
onStart	null	String	Client side callback to execute when resizing begins.
onResize	null	String	Client side callback to execute during resizing.
onStop	null	String	Client side callback to execute after resizing end.

Getting started with Resizable

Resizable is used by setting *for* option as the identifier of the target.

```
<p:graphicImage id="img" value="campnou.jpg" />

<p:resizable for="img" />
```

Another example is the input fields, if users need more space for a textarea, make it resizable by;

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area" />
```

Boundaries

To prevent overlapping with other elements on page, boundaries need to be specified. There're 4 attributes for this *minWidth*, *maxWidth*, *minHeight* and *maxHeight*. The valid values for these attributes are numbers in terms of pixels.

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area" minWidth="20" minHeight="40" maxWidth="50" maxHeight="100"/>
```

Handles

Resize handles to display are customize using *handles* attribute with a combination of *n*, *e*, *s*, *w*, *ne*, *se*, *sw* and *nw* as the value. Default value is "*e*, *s*, *se*".

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area" handles="e,w,n,se,sw,ne,nw"/>
```

Visual Feedback

Resize helper is the element used to provide visual feedback during resizing. By default actual element itself is the helper and two options are available to customize the way feedback is provided. Enabling *ghost* option displays the element itself with a lower opacity, in addition enabling *proxy* option adds a css class called *.ui-resizable-proxy* which you can override to customize.

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area" proxy="true" />
```

```
.ui-resizable-proxy {
     border: 2px dotted #00F;
}
```

Effects

Resizing can be animated using *animate* option and setting an *effect* name. Animation speed is customized using *effectDuration* option *"slow"*, *"normal"* and *"fast"* as valid values.

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area" animate="true" effect="swing" effectDuration="normal" />
```

Following is the list of available effect names;

• swing • easeInQuad • easeOutQuad • easeInOutQuad • easeInCubic • easeOutCubic • easeInOutCubic	• easeInQuart • easeOutQuart • easeInOutQuart • easeInQuint • easeOutQuint • easeInOutQuint • easeInSine	• easeOutSine • easeInExpo • easeOutExpo • easeInOutExpo • easeInCirc • easeOutCirc • easeInOutCirc	• easeInElastic • easeOutElastic • easeInOutElastic • easeInBack • easeOutBack • easeInOutBack	• easeInBounce • easeOutBounce • easeInOutBounce

Ajax Behavior Events

Resizable provides default and only *resize* event that is called on resize end. In case you have a listener defined, it will be called by passing an *org.primefaces.event.ResizeEvent* instance as a parameter.

```
<h:inputTextarea id="area" value="Resize me if you need more space" />

<p:resizable for="area">
     <p:ajax listener="#{resizeBean.handleResize}">
 </p:resizable>
```

```
public class ResizeBean {

      public void handleResize(ResizeEvent event) {
            int width = event.getWidth();
            int height = event.getHeight();
      }
}
```

Client Side Callbacks

Resizable has three client side callbacks you can use to hook-in your javascript; *onStart*, *onResize* and *onStop*. All of these callbacks receive two parameters that provide various information about resize event.

```
<h:inputTextarea id="area" value="Resize me if you need more space" />
<p:resizable for="area" onStop="handleStop(event, ui)" />
```

```
function handleStop(event, ui) {
      //ui.helper = helper element as a jQuery object
      //ui.originalPosition = top, left position before resizing
      //ui.originalSize = width, height before resizing
      //ui.positon = top, left after resizing
      //ui.size = width height of current size
}
```

Skinning

Style Class	Applies
.ui-resizable	Element that is resizable
.ui-resizable-handle	Handle element
.ui-resizable-handle-{handlekey}	Particular handle element identified by handlekey like e, s, ne
.ui-resizable-proxy	Proxy helper element for visual feedback

3.83 Ring

Ring is a data display component with a circular animation.

Info

Tag	**ring**
Component Class	**org.primefaces.component.ring.Ring**
Component Type	**org.primefaces.component.Ring**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RingRenderer**
Renderer Class	**org.primefaces.component.ring.RingRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
value	null	Object	Collection to display.
var	null	String	Name of the data iterator.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
easing	swing	String	Type of easing to use in animation.

Getting started with Ring

A collection is required to use the Ring component.

```
public class RingBean {

      private List<Player> players;

      public RingBean() {
            players = new ArrayList<Player>();

            players.add(new Player("Messi", 10, "messi.jpg", "CF"));
            players.add(new Player("Iniesta", 8, "iniesta.jpg", "CM"));
            players.add(new Player("Villa", 7, "villa.jpg", "CF"));
            players.add(new Player("Xavi", 6, "xavi.jpg", "CM"));
            players.add(new Player("Puyol", 5, "puyol.jpg", "CB"));
      }

      //getter&setter for players
}
```

```
<p:ring value="#{ringBean.players}" var="player”>
      <p:graphicImage value="/images/barca/#{player.photo}"/>
</p:ring>
```

Item Selection

A column is required to process item selection from ring properly.

```
<p:ring value="#{ringBean.players}" var="player">
      <p:column>
            //UI to select an item e.g. commandLink
      </p:column>
</p:ring>
```

Easing

Following is the list of available options for easing animation.

• swing • easeInQuad • easeOutQuad • easeInOutQuad • easeInCubic • easeOutCubic • easeInOutCubic	• easeInQuart • easeOutQuart • easeInOutQuart • easeInQuint • easeOutQuint • easeInOutQuint • easeInSine	• easeOutSine • easeInExpo • easeOutExpo • easeInOutExpo • easeInCirc • easeOutCirc • easeInOutCirc	• easeInElastic • easeOutElastic • easeInOutElastic • easeInBack • easeOutBack • easeInOutBack	• easeInBounce • easeOutBounce • easeInOutBounce

Skinning

Ring resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes.

Style Class	Applies
.ui-ring	Main container element.
.ui-ring-item	Each item in the list.

3.84 Row

Row is a helper component for datatable.

Info

Tag	**row**
Component Class	**org.primefaces.component.row.Row**
Component Type	**org.primefaces.component.Row**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting Started with Row

See datatable grouping section for more information about how row is used.

3.85 RowEditor

RowEditor is a helper component for datatable.

Info

Tag	**rowEditor**
Component Class	**org.primefaces.component.roweditor.RowEditor**
Component Type	**org.primefaces.component.RowEditor**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RowEditorRenderer**
Renderer Class	**org.primefaces.component.roweditor.RowEditorRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting Started with RowEditor

See inline editing section in datatable documentation for more information about usage.

3.86 RowExpansion

RowExpansion is a helper component of datatable used to implement expandable rows.

Info

Tag	**rowExpansion**
Component Class	**org.primefaces.component.rowexpansion.RowExpansion**
Component Type	**org.primefaces.component.RowExpansion**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting Started with RowExpansion

See datatable expandable rows section for more information about how rowExpansion is used.

3.87 RowToggler

RowToggler is a helper component for datatable.

Info

Tag	**rowToggler**
Component Class	**org.primefaces.component.rowtoggler.RowToggler**
Component Type	**org.primefaces.component.RowToggler**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.RowTogglerRenderer**
Renderer Class	**org.primefaces.component.rowtoggler.RowTogglerRenderer**

Attributes

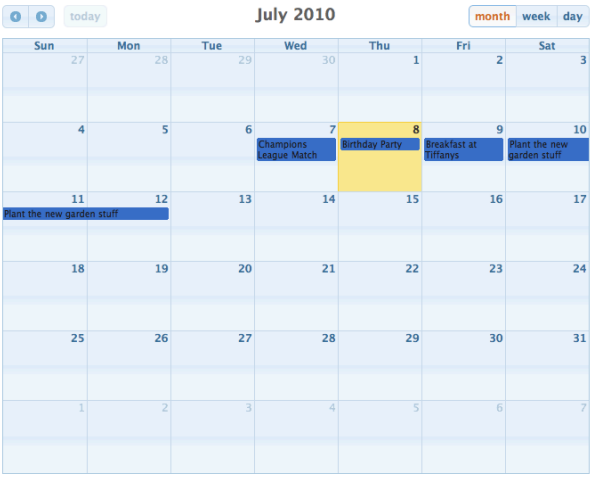
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean

Getting Started with Row

See expandable rows section in datatable documentation for more information about usage.

3.88 Schedule

Schedule provides an Outlook Calendar, iCal like JSF component to manage events.



Info

Tag	**schedule**
Component Class	**org.primefaces.component.schedule.Schedule**
Component Type	**org.primefaces.component.Schedule**
Component Family	**org.primefaces**
Renderer Type	**org.primefaces.component.ScheduleRenderer**
Renderer Class	**org.primefaces.component.schedule.ScheduleRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
value	null	Object	An org.primefaces.model.ScheduleModel instance representing the backed model
locale	null	Object	Locale for localization, can be String or a java.util.Locale instance

Name	Default	Type	Description
aspectRatio	null	Float	Ratio of calendar width to height, higher the value shorter the height is
view	month	String	The view type to use, possible values are 'month', 'agendaDay', 'agendaWeek', 'basicWeek', 'basicDay'
initialDate	null	Object	The initial date that is used when schedule loads. If ommitted, the schedule starts on the current date
showWeekends	TRUE	Boolean	Specifies inclusion Saturday/Sunday columns in any of the views
style	null	String	Style of the main container element of schedule
styleClass	null	String	Style class of the main container element of schedule
draggable	TRUE	Boolean	When true, events are draggable.
resizable	TRUE	Boolean	When true, events are resizable.
showHeader	TRUE	Boolean	Specifies visibility of header content.
leftHeaderTemplate	prev, next today	String	Content of left side of header.
centerHeaderTemplate	title	String	Content of center of header.
rightHeaderTemplate	month, agendaWeek, agendaDay	String	Content of right side of header.
allDaySlot	TRUE	Boolean	Determines if all-day slot will be displayed in agendaWeek or agendaDay views
slotMinutes	30	Integer	Interval in minutes in an hour to create a slot.
firstHour	6	Integer	First hour to display in day view.
minTime	null	String	Minimum time to display in a day view.
maxTime	null	String	Maximum time to display in a day view.
axisFormat	null	String	Determines the time-text that will be displayed on the vertical axis of the agenda views.
timeFormat	null	String	Determines the time-text that will be displayed on each event.
timeZone	null	Object	String or a java.util.TimeZone instance to specify the timezone used for date conversion.

Getting started with Schedule

Schedule needs to be backed by an *org.primefaces.model.ScheduleModel* instance, a schedule model consists of *org.primefaces.model.ScheduleEvent* instances.

```
<p:schedule value="#{scheduleBean.model}" />
```

```
public class ScheduleBean {

      private ScheduleModel model;

      public ScheduleBean() {
            eventModel = new ScheduleModel<ScheduleEvent>();
            eventModel.addEvent(new DefaultScheduleEvent("title", new Date(),
                                    new Date()));
      }

      public ScheduleModel getModel() { return model; }
}
```

DefaultScheduleEvent is the default implementation of ScheduleEvent interface. Mandatory properties required to create a new event are the title, start date and end date. Other properties such as allDay get sensible default values. Table below describes each property in detail.

Property	Description
id	Used internally by PrimeFaces, auto generated.
title	Title of the event.
startDate	Start date of type java.util.Date.
endDate	End date of type java.util.Date.
allDay	Flag indicating event is all day.
styleClass	Visual style class to enable multi resource display.
data	Optional data you can set to be represented by Event.
editable	Whether the event is editable or not.

Ajax Behavior Events

Schedule provides various ajax behavior events to respond user actions.

Event	Listener Parameter	Fired
dateSelect	org.primefaces.event.DateSelectEvent	When a date is selected.
eventSelect	org.primefaces.event.ScheduleEntrySelectEvent	When an event is selected.

Event	Listener Parameter	Fired
eventMove	org.primefaces.event.ScheduleEntryMoveEvent	When an event is moved.
eventResize	org.primefaces.event.ScheduleEntryResizeEvent	When an event is resized.

Ajax Updates

Schedule has a quite complex UI which is generated on-the-fly by the client side PrimeFaces.widget.Schedule widget to save bandwidth and increase page load performance. As a result when you try to update schedule like with a regular PrimeFacess PPR, you may notice a UI lag as the DOM will be regenerated and replaced. Instead, Schedule provides a simple client side api and the *update* method. Whenever you call update, schedule will query it's server side ScheduleModel instance to check for updates, transport method used to load events dynamically is JSON, as a result this approach is much more effective then updating with regular PPR. An example of this is demonstrated at editable schedule example, save button is calling *myschedule.update()* at oncomplete event handler.

Editable Schedule

Let's put it altogether to come up a fully editable and complex schedule.

```
<h:form>
      <p:schedule value="#{bean.eventModel}" editable="true" widgetVar="myschedule">
        <p:ajax event="dateSelect" listener="#{bean.onDateSelect}"
                          update="eventDetails" oncomplete="eventDialog.show()" />
        <p:ajax event="eventSelect" listener="#{bean.onEventSelect}"
       </p:schedule>

      <p:dialog widgetVar="eventDialog" header="Event Details">
       <h:panelGrid id="eventDetails" columns="2">
        <h:outputLabel for="title" value="Title:" />
        <h:inputText id="title" value="#{bean.event.title}" required="true"/>

        <h:outputLabel for="from" value="From:" />
        <p:inputMask id="from" value="#{bean.event.startDate}" mask="99/99/9999">
          <f:convertDateTime pattern="dd/MM/yyyy" />
         </p:inputMask>

        <h:outputLabel for="to" value="To:" />
          <p:inputMask id="to" value="#{bean.event.endDate}" mask="99/99/9999">
             <f:convertDateTime pattern="dd/MM/yyyy" />
          </p:inputMask>

          <h:outputLabel for="allDay" value="All Day:" />
          <h:selectBooleanCheckbox id="allDay" value="#{bean.event.allDay}" />

          <p:commandButton type="reset" value="Reset" />
          <p:commandButton value="Save" actionListener="#{bean.addEvent}"
                    oncomplete="myschedule.update();eventDialog.hide();"/>
      </h:panelGrid>
     </p:dialog>
</h:form>
```

```
public class ScheduleBean {

      private ScheduleModel<ScheduleEvent> model;
      private ScheduleEventImpl event = new DefaultScheduleEvent();

      public ScheduleBean() {
            eventModel = new ScheduleModel<ScheduleEvent>();
      }

      public ScheduleModel<ScheduleEvent> getModel() { return model; }

      public ScheduleEventImpl getEvent() { return event; }

      public void setEvent(ScheduleEventImpl event) { this.event = event; }

      public void addEvent() {
            if(event.getId() == null)
                  eventModel.addEvent(event);
            else
                  eventModel.updateEvent(event);

            event = new DefaultScheduleEvent();    //reset dialog form
      }

      public void onEventSelect(ScheduleEntrySelectEvent e) {
            event = e.getScheduleEvent();
      }

      public void onDateSelect(DateSelectEvent e) {
            event = new DefaultScheduleEvent("", e.getDate(), e.getDate());
      }
}
```

Lazy Loading

Schedule assumes whole set of events are eagerly provided in ScheduleModel, if you have a huge data set of events, lazy loading features would help to improve performance.

In lazy loading mode, only the events that belong to the displayed time frame are fetched whereas in default eager more all events need to be loaded.

```
<p:schedule value="#{scheduleBean.lazyModel}" />
```

To enable lazy loading of Schedule events, you just need to provide an instance of *org.primefaces.model.LazyScheduleModel* and implement the *loadEvents* methods. *loadEvents* method is called with new boundaries every time displayed timeframe is changed.

```
public class ScheduleBean {

      private ScheduleModel lazyModel;

      public ScheduleBean() {

            lazyModel = new LazyScheduleModel() {

                  @Override
                  public void loadEvents(Date start, Date end) {
                        //addEvent(...);
                        //addEvent(...);
                  }
            };
      }

      public ScheduleModel getLazyModel() {
            return lazyModel;
      }
}
```

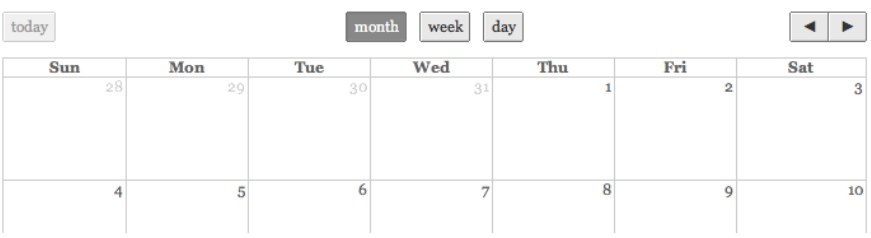
Customizing Header

Header controls of Schedule can be customized based on templates, valid values of template options are;

- title: Text of current month/week/day information
- prev: Button to move calendar back one month/week/day.
- next: Button to move calendar forward one month/week/day.
- prevYear: Button to move calendar back one year
- nextYear: Button to move calendar forward one year
- today: Button to move calendar to current month/week/day.
- *viewName*: Button to change the view type based on the view type.

These controls can be placed at three locations on header which are defined with *leftHeaderTemplate*, *rightHeaderTemplate* and *centerTemplate* attributes.

```
<p:schedule value="#{scheduleBean.model}"
      leftHeaderTemplate"today"
      rightHeaderTemplate="prev,next"
      centerTemplate="month, agendaWeek, agendaDay"
</p:schedule>
```

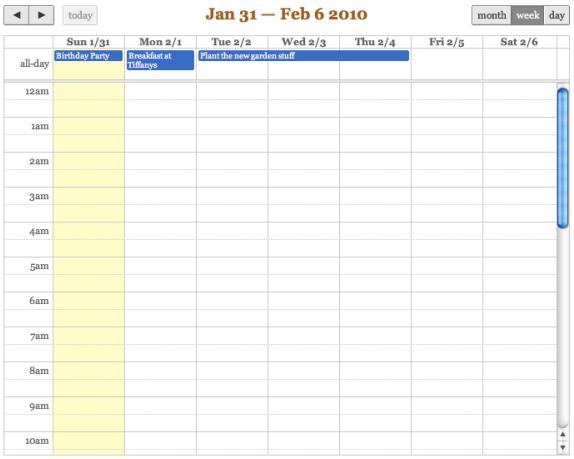


Views

5 different views are supported, these are "month", "agendaWeek", "agendaDay", "basicWeek" and "basicDay".

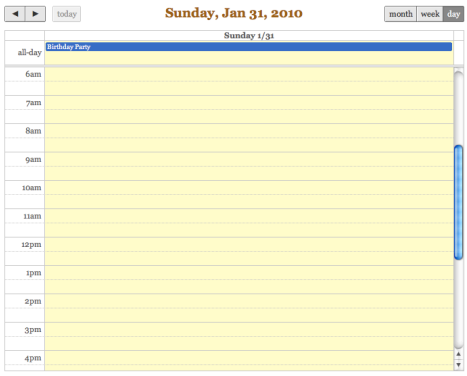
agendaWeek

```
<p:schedule value="#{scheduleBean.model}" view="agendaWeek"/>
```



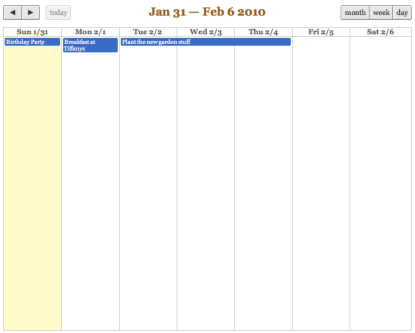
agendaDay

```
<p:schedule value="#{scheduleBean.model}" view="agendaDay"/>
```



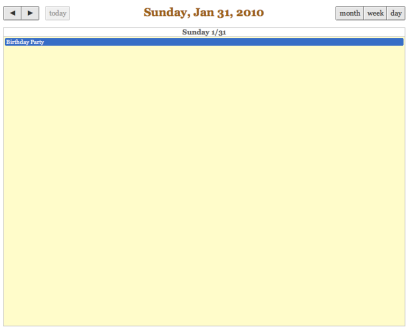
basicWeek

```
<p:schedule value="#{scheduleBean.model}" view="basicWeek"/>
```



basicDay

```
<p:schedule value="#{scheduleBean.model}" view="basicDay"/>
```



Locale Support

By default locale information is retrieved from the view's locale and can be overridden by the locale attribute. Locale attribute can take a locale key as a String or a java.util.Locale instance. Default language of labels are English and you need to add the necessary translations to your page manually as PrimeFaces does not include language translations. PrimeFaces Wiki Page for PrimeFacesLocales is a community driven page where you may find the translations you need. Please contribute to this wiki with your own translations.

```
http://wiki.primefaces.org/display/Components/PrimeFaces+Locales
```

Translation is a simple javascript object, we suggest adding the code to a javascript file and include in your application. Following is a Turkish calendar.

```
<p:schedule value="#{scheduleBean.model}" locale="tr"/>
```

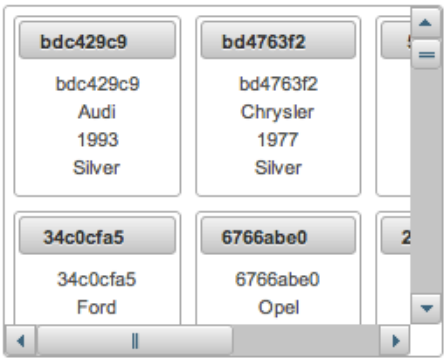
Skinning

Schedule resides in a main container which *style* and *styleClass* attributes apply.

As skinning style classes are global, see the main skinning section for more information.

3.89 ScrollPanel

ScrollPanel is used to display overflowed content with theme aware scrollbars instead of native browsers scrollbars.



Info

Tag	**scrollPanel**
Component Class	**org.primefaces.component.scrollpanel.ScrollPanel**
Component Type	**org.primefaces.component.ScrollPanel**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ScrollPanelRenderer**
Renderer Class	**org.primefaces.component.scrollpanel.ScrollPanelRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
mode	default	String	Scrollbar display mode, valid values are default and native.

Getting started with ScrollPanel

ScrollPanel is used a container component, width and height must be defined.

```
<p:scrollPanel style="width:250px;height:200px">
      //any content
</p:scrollPanel>
```

Native ScrollBars

By default, scrollPanel displays theme aware scrollbars, setting mode option to native displays browser scrollbars.

```
<p:scrollPanel style="width:250px;height:200px" mode="native">
      //any content
</p:scrollPanel>
```

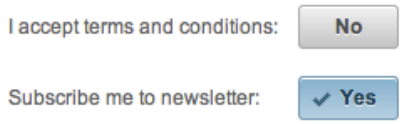
Skinning

ScrollPanel resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-scrollpanel	Main container element.
.ui-scrollpanel-container	Overflow container.
.ui-scrollpanel-hbar	Horizontal scrollbar.
.ui-scrollpanel-vbar	Vertical scrollbar.
.ui-scrollpanel-handle	Handle of a scrollbar

3.90 SelectBooleanButton

SelectBooleanButton is used to select a binary decision with a toggle button.



Info

Tag	**selectBooleanButton**
Component Class	**org.primefaces.component.selectbooleanbutton.SelectBooleanButton**
Component Type	**org.primefaces.component.SelectBooleanButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectBooleanButtonRenderer**
Renderer Class	**org.primefaces.component.selectbooleanbutton.SelectBooleanButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent

Name	Default	Type	Description
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.
onLabel	null	String	Label to display when button is selected.
offLabel	null	String	Label to display when button is unselected.
onIcon	null	String	Icon to display when button is selected.
offIcon	null	String	Icon to display when button is unselected.

Getting started with SelectBooleanButton

SelectBooleanButton usage is similar to selectBooleanCheckbox.

```
<p:selectBooleanButton id="value2" value="#{bean.value}" onLabel="Yes"
      offLabel="No" onIcon="ui-icon-check" offIcon="ui-icon-close" />
```

```
public class Bean {
      private boolean value;
      //getter and setter
}
```

Skinning

SelectBooleanButton resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectbooleanbutton	Main container element.

3.91 SelectBooleanCheckbox

SelectBooleanCheckbox is an extended version of the standard checkbox with theme integration.

Info

Tag	**selectBooleanCheckbox**
Component Class	**org.primefaces.component.selectbooleancheckbox.SelectBooleanCheckbox**
Component Type	**org.primefaces.component.SelectBooleanCheckbox**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectBooleanCheckboxRenderer**
Renderer Class	**org.primefaces.component.selectbooleancheckbox.SelectBooleanCheckbox Renderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.

Name	Default	Type	Description
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.
itemLabel	null	String	Label displayed next to checkbox.

Getting started with SelectBooleanCheckbox

SelectBooleanCheckbox usage is same as the standard one.

Client Side API

Widget: *PrimeFaces.widget.SelectBooleanCheckbox*

Method	Params	Return Type	Description
check()	-	void	Checks the checkbox.
uncheck()	-	void	Unchecks the checkbox.
toggle()	-	void	Toggles check state.

Skinning

SelectBooleanCheckbox resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-chkbox	Main container element.
.ui-chkbox-box	Container of checkbox icon.
.ui-chkbox-icon	Checkbox icon.
.ui-chkbox-icon	Checkbox label.

3.92 SelectChecboxMenu

SelectCheckboxMenu is a multi select component that displays options in an overlay.

Info

Tag	**selectCheckboxMenu**
Component Class	**org.primefaces.component.selectcheckboxmenu.SelectCheckboxMenu**
Component Type	**org.primefaces.component.SelectCheckboxMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectCheckboxMenuRenderer**
Renderer Class	**org.primefaces.component.selectcheckboxmenu.SelectCheckboxMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent

Name	Default	Type	Description
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.
height	null	Integer	Height of the overlay.

Getting started with SelectCheckboxMenu

SelectCheckboxMenu usage is same as the standard selectManyCheckbox or PrimeFaces selectCheckboxMenu components.

Skinning

SelectCheckboxMenu resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectcheckboxmenu	Main container element.
.ui-selectcheckboxmenu-label-container	Label container.
.ui-selectcheckboxmenu-label	Label.
.ui-selectcheckboxmenu-trigger	Dropdown icon.
.ui-selectcheckboxmenu-panel	Overlay panel.
.ui-selectcheckboxmenu-items	Option list container.
.ui-selectcheckboxmenu-item	Each options in the collection.
.ui-chkbox	Container of a checkbox.
.ui-chkbox-box	Container of checkbox icon.
.ui-chkbox-icon	Checkbox icon.

3.93 SelectManyButton

SelectManyButton is a multi select component using button UI.

Info

Tag	**selectManyButton**
Component Class	**org.primefaces.component.selectmanybutton.SelectManyButton**
Component Type	**org.primefaces.component.SelectManyButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectManyButton**
Renderer Class	**org.primefaces.component.selectmanybutton.SelectManyButton**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent

Name	Default	Type	Description
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectManyButton

SelectManyButton usage is same as selectManyCheckbox, buttons just replace checkboxes.

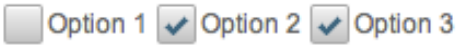
Skinning

SelectManyButton resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectmanybutton	Main container element.

3.94 SelectManyCheckbox

SelectManyCheckbox is an extended version of the standard SelectManyCheckbox with theme integration.



Info

Tag	**selectManyCheckbox**
Component Class	**org.primefaces.component.selectmanycheckbox.SelectManyCheckbox**
Component Type	**org.primefaces.component.SelectManyCheckbox**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectManyCheckboxRenderer**
Renderer Class	**org.primefaces.component.selectmanycheckbox.SelectManyCheckboxRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent

Name	Default	Type	Description
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
layout	lineDirection	String	Layout of the checkboxes, valid values are "lineDirection"(horizontal) and "pageDirection"(vertical).
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectManyCheckbox

SelectManyCheckbox usage is same as the standard one.

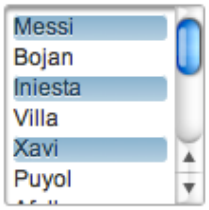
Skinning

SelectManyCheckbox resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectmanycheckbox	Main container element.
.ui-chkbox	Container of a checkbox.
.ui-chkbox-box	Container of checkbox icon.
.ui-chkbox-icon	Checkbox icon.

3.95 SelectManyMenu

SelectManyMenu is an extended version of the standard SelectManyMenu with theme integration.



Info

Tag	**selectManyMenu**
Component Class	**org.primefaces.component.selectmanymenu.SelectManyMenu**
Component Type	**org.primefaces.component.SelectManyMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectManyMenuRenderer**
Renderer Class	**org.primefaces.component.selectmanymenu.SelectManyMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input

Name	Default	Type	Description
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectManyMenu

SelectManyMenu usage is same as the standard one.

Skinning

SelectManyMenu resides in a container that *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectmanymenu	Main container element.
.ui-selectlistbox-item	Each item in list.

3.96 SelectOneButton

SelectOneButton is an input component to do a single select.

Info

Tag	**selectOneButton**
Component Class	**org.primefaces.component.selectonebutton.SelectOneButton**
Component Type	**org.primefaces.component.SelectOneButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectOneButtonRenderer**
Renderer Class	**org.primefaces.component.selectonebutton.SelectOneButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent

Name	Default	Type	Description
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectOneButton

SelectOneButton usage is same as selectOneRadio component, buttons just replace the radios.

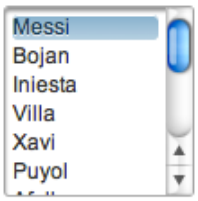
Skinning

SelectOneButton resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectonebutton	Main container element.

3.97 SelectOneListbox

SelectOneListbox is an extended version of the standard SelectOneListbox with theme integration.



Info

Tag	**selectOneListbox**
Component Class	**org.primefaces.component.selectonelistbox.SelectOneListbox**
Component Type	**org.primefaces.component.SelectOneListbox**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectOneListboxRenderer**
Renderer Class	**org.primefaces.component.selectonelistbox.SelectOneListBoxRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input

Name	Default	Type	Description
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectOneListbox

SelectOneListbox usage is same as the standard one.

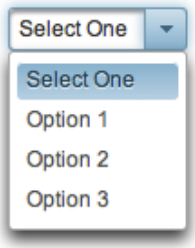
Skinning

SelectOneListbox resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectonelistbox	Main container element.
.ui-selectlistbox-item	Each item in list.

3.98 SelectOneMenu

SelectOneMenu is an extended version of the standard SelectOneMenu with theme integration.



Info

Tag	**selectOneMenu**
Component Class	**org.primefaces.component.selectonemenu.SelectOneMenu**
Component Type	**org.primefaces.component.SelectOneMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectOneMenuRenderer**
Renderer Class	**org.primefaces.component.selectonemenu.SelectOneMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required

Name	Default	Type	Description
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
effect	blind	String	Name of the toggle animation.
effectDuration	400	Integer	Duration of toggle animation in milliseconds.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Client side callback to execute on value change.
onkeyup	null	String	Client side callback to execute on keyup.
onkeydown	null	String	Client side callback to execute on keydown.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.
var	null	String	Name of the item iterator.
height	auto	Integer	Height of the overlay.
tabindex	null	String	Tabindex of the input.
editable	FALSE	Boolean	When true, input becomes editable.

Getting started with SelectOneMenu

Basic SelectOneMenu usage is same as the standard one.

Effects

An animation is executed to show and hide the overlay menu, default effect is blind and following options are available for *effect* attribute;

- blind
- bounce
- clip
- drop

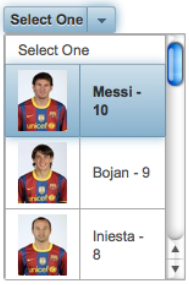
- explode
- fold
- highlight
- puff
- pulsate
- scale
- shake
- size
- slide

Custom Content

SelectOneMenu can display custom content in overlay panel by using column component and the var option to refer to each item.

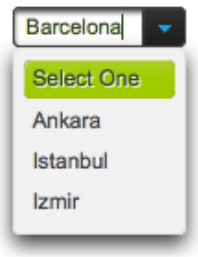
```
public class MenuBean {
      private List<Player> players;
      private Player selectedPlayer;

      public OrderListBean() {
            players = new ArrayList<Player>();

            players.add(new Player("Messi", 10, "messi.jpg"));
            //more players
      }

      //getters and setters
}
```

```
<p:selectOneMenu value="#{menuBean.selectedPlayer}" converter="player" var="p">
     <f:selectItem itemLabel="Select One" itemValue="" />
     <f:selectItems value="#{menuBean.players}" var="player"
                  itemLabel="#{player.name}" itemValue="#{player}"/>
      <p:column>
        <p:graphicImage value="/images/barca/#{p.photo}" width="40" height="50"/>
      </p:column>

      <p:column>
            #{p.name} - #{p.number}
      </p:column>
</p:selectOneMenu>
```



Editable

Editable SelectOneMenu provides a UI to either choose from the predefined options or enter a manual input. Set editable option to true to use this feature.



Client Side API

Widget: *PrimeFaces.widget.SelectOneMenu*

Method	Params	Return Type	Description
show()	-	void	Shows overlay menu.
hide()	-	void	Hides overlay menu.
blur()	-	void	Invokes blur event.
focus()	-	void	Invokes focus event.
enable()	-	void	Enables component.
disable()	-	void	Disabled component.
selectValue()	value: itemValue	void	Selects given value.

Skinning

SelectOneMenu resides in a container element that *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectonemenu	Main container.
.ui-selectonemenu-label	Label of the component.
.ui-selectonemenu-trigger	Container of dropdown icon.
.ui-selectonemenu-items	Items list.
.ui-selectonemenu-items	Each item in the list.

3.99 SelectOneRadio

SelectOneRadio is an extended version of the standard SelectOneRadio with theme integration.

Info

Tag	**selectOneRadio**
Component Class	**org.primefaces.component.selectoneradio.SelectOneRadio**
Component Type	**org.primefaces.component.SelectOneRadio**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SelectOneRadioRenderer**
Renderer Class	**org.primefaces.component.selectoneradio.SelectOneRadioRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component referring to a List.
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	When set true, process validations logic is executed at apply request values phase for this component.
required	FALSE	Boolean	Marks component as required
validator	null	MethodExpr	A method expression that refers to a method validationg the input
valueChangeListener	null	MethodExpr	A method expression that refers to a method for handling a valuechangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.

Name	Default	Type	Description
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
layout	lineDirection	String	Layout of the checkboxes, valid values are "lineDirection"(horizontal) and "pageDirection"(vertical).
onchange	null	String	Callback to execute on value change.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the container.

Getting started with SelectOneRadio

SelectOneRadio usage is same as the standard one.

Custom Layout

Standard selectOneRadio component only supports horizontal and vertical rendering of the radio buttons with a strict table markup. PrimeFaces SelectOneRadio on the other hand provides a flexible layout option so that radio buttons can be located anywhere on the page. This is implemented by setting layout option to custom and with standalone radioButton components. Note that in custom mode, selectOneRadio itself does not render any output.

```
 <p:selectOneRadio id="customRadio" value="#{formBean.option}" layout="custom">
      <f:selectItem itemLabel="Option 1" itemValue="1" />
      <f:selectItem itemLabel="Option 2" itemValue="2" />
      <f:selectItem itemLabel="Option 3" itemValue="3" />
</p:selectOneRadio>

<h:panelGrid columns="3">
      <p:radioButton id="opt1" for="customRadio" itemIndex="0"/>
      <h:outputLabel for="opt1" value="Option 1" />
      <p:spinner />

      <p:radioButton id="opt2" for="customRadio" itemIndex="1"/>
      <h:outputLabel for="opt2" value="Option 2" />
      <p:inputText />

      <p:radioButton id="opt3" for="customRadio" itemIndex="2"/>
      <h:outputLabel for="opt3" value="Option 3" />
      <p:calendar />
</h:panelGrid>
```

RadioButton's for attribute should refer to a selectOneRadio component and itemIndex points to the index of the selectItem. When using custom layout option, selectOneRadio component should be placed above any radioButton that points to the selectOneRadio.

Skinning

SelectOneRadio resides in a main container which *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-selectoneradio	Main container element.
.ui-radiobutton	Container of a radio button.
.ui-radiobutton-box	Container of radio button icon.
.ui-radiobutton-icon	Radio button icon.

3.100 Separator

Seperator displays a horizontal line to separate content.

Info

Tag	**separator**
Component Class	**org.primefaces.component.separator.Separator**
Component Type	**org.primefaces.component.Separator**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.Separator**
Renderer Class	**org.primefaces.component.separator.Separator**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
title	null	String	Advisory tooltip informaton.
style	null	String	Inline style of the separator.
styleClass	null	String	Style class of the separator.

Getting started with Separator

In its simplest form, separator is used as;

```
//content

<p:separator />

//content
```

Dimensions

Separator renders a <*hr* /> tag which style and styleClass options apply.

```
<p:separator style="width:500px;height:20px" />
```

Special Separators

Separator can be used inside other components such as menu and toolbar as well.

```
<p:menu>
      //submenu or menuitem
      <p:separator />
      //submenu or menuitem
</p:menu>

<p:toolbar>
      <p:toolbarGroup align="left">
            //content
            <p:separator />
            //content
      </p:toolbarGroup>
</p:toolbar>
```

Skinning

As mentioned in dimensions section, style and styleClass options can be used to style the separator.

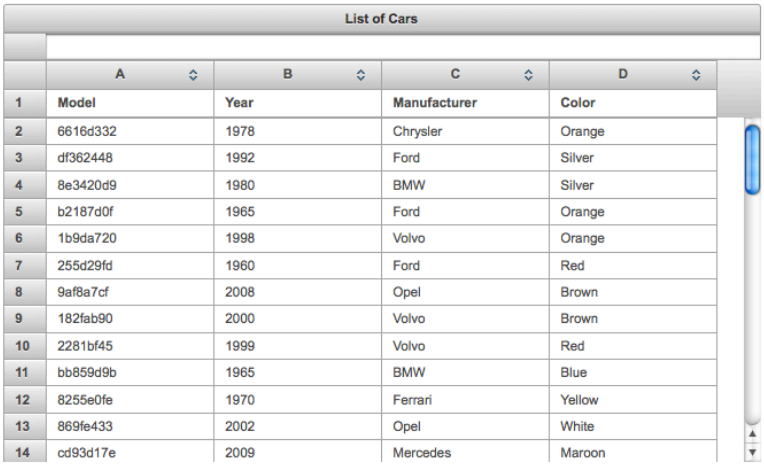
Following is the list of structural style classes;

Class	Applies
.ui-separator	Separator element

As skinning style classes are global, see the main Skinning section for more information.

3.101 Sheet

Sheet is an excel-like component to do data manipulation featuring resizable columns, ajax sorting, horizontal/vertical scrolling, frozen headers, keyboard navigation, multi cell selection with meta/shift keys, bulk delete/update and more.


List of Cars
A
B
C
D
1 Model Year Manufacturer Color
2 6616d332 1978 Chrysler Orange
3 df362448 1992 Ford Silver
4 8e3420d9 1980 BMW Silver
5 b2187d0f 1965 Ford Orange
6 1b9da720 1998 Volvo Orange
7 255d29fd 1960 Ford Red
8 9af8a7cf 2008 Opel Brown
9 182fab90 2000 Volvo Brown
10 2281bf45 1999 Volvo Red
11 bb859d9b 1965 BMW Blue
12 8255e0fe 1970 Ferrari Yellow
13 869fe433 2002 Opel White
14 cd93d17e 2009 Mercedes Maroon

Info

Tag	**sheet**
Component Class	**org.primefaces.component.sheet.Sheet**
Component Type	**org.primefaces.component.Sheet**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SheetRenderer**
Renderer Class	**org.primefaces.component.sheet.SheetRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data of the component.
var	null	String	Name of the data iterator.
widgetVar	null	String	Name of the client side widget.
style	null	String	Inline style of the container element.

Name	Default	Type	Description
styleClass	null	String	Style class of the container element.
scrollWidth	null	Integer	Width of the viewport.
scrollHeight	null	Integer	Height of the viewport.

Getting started with Sheet

Sheet usage is similar to a datatable, two important points for sheet are;

- Columns must be fixed width.
- Column child must be an input text.

```
public class TableBean {

      private List<Car> cars;

      public TableBean() {
            cars = //populate data
      }

      //getters and setters
}
```

```
<p:sheet value="#{tableBean.cars}" var="car" scrollHeight="300">
      <f:facet name="caption">
            List of Cars
      </f:facet>

      <p:column headerText="Model" style="width:125px">
            <h:inputText value="#{car.model}" />
      </p:column>

      <p:column headerText="Year" style="width:125px">
            <h:inputText value="#{car.year}"/>
      </p:column>

      <p:column headerText="Manufacturer" style="width:125px">
            <h:inputText value="#{car.manufacturer}" />
      </p:column>

      <p:column headerText="Color" style="width:125px">
            <h:inputText value="#{car.color}" />
      </p:column>
</p:sheet>
```

When the parent form of the sheet is submitted, sheet will update the data according to the changes on client side.

Sorting

Sorting can be enabled using the sortBy option of the column component.

```
<p:column headerText="Model" style="width:125px" sortBy=”#{car.model}”>
      <h:inputText value="#{car.model}" />
</p:column>
```

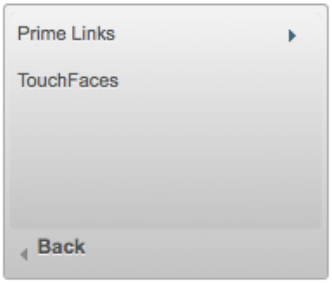
Skinning

Sheet resides in a container that *style* and *styleClass* attributes apply. As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes;

Style Class	Applies
.ui-sheet	Main container element.
.ui-sheet-editor-bar	Editor bar.
.ui-sheet-cell-info	Label to display current cell.
.ui-sheet-editor	Global cell editor.
.ui-sh-c	Each cell.
.ui-sh-c-d	Label of a cell.
.ui-sh-c-e	Editor of a cell.

3.102 SlideMenu

TieredMenu is used to display nested submenus with sliding animation.



Info

Tag	**slideMenu**
Component Class	**org.primefaces.component.slidemenu.SlideMenu**
Component Type	**org.primefaces.component.SlideMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SlideMenuRenderer**
Renderer Class	**org.primefaces.component.slidemenu.SlideMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
model	null	org.primefaces.model.MenuModel	MenuModel instance for programmatic menu.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
backLabel	Back	String	Text for back link.
trigger	null	String	Id of the component whose triggerEvent will show the dynamic positioned menu.

Name	Default	Type	Description
my	null	String	Corner of menu to align with trigger element.
at	null	String	Corner of trigger to align with menu element.
overlay	FALSE	Boolean	Defines positioning, when enabled menu is displayed with absolute position relative to the trigger. Default is false, meaning static positioning.
triggerEvent	click	String	Event name of trigger that will show the dynamic positioned menu.

Getting started with the SlideMenu

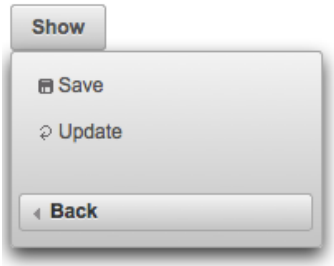
SlideMenu consists of submenus and menuitems, submenus can be nested and each nested submenu will be displayed with a slide animation.

```
<p:slideMenu>
   <p:submenu label="Ajax Menuitems" icon="ui-icon-refresh">
       <p:menuitem value="Save" actionListener="#{buttonBean.save}"
                        update="messages" icon="ui-icon-disk" />
      <p:menuitem value="Update" actionListener="#{buttonBean.update}"
                        update="messages" icon="ui-icon-arrowrefresh-1-w" />
      </p:submenu>

      <p:submenu label="Non-Ajax Menuitem" icon="ui-icon-newwin">
            <p:menuitem value="Delete" actionListener="#{buttonBean.delete}"
                  update="messages" ajax="false" icon="ui-icon-close"/>
      </p:submenu>

      <p:separator />

      <p:submenu label="Navigations" icon="ui-icon-extlink">
            <p:submenu label="Prime Links">
                  <p:menuitem value="Prime" url="http://www.prime.com.tr" />
                  <p:menuitem value="PrimeFaces" url="http://www.primefaces.org" />
            </p:submenu>
            <p:menuitem value="Mobile" url="/mobile" />
      </p:submenu>
</p:slideMenu>
```

Overlay

SlideMenu can be positioned relative to a trigger component, following sample attaches a slideMenu to the button so that whenever the button is clicked menu will be displayed in an overlay itself.

```
<p:commandButton type="button" value="Show" id="btn" />

<p:slideMenu trigger="btn" my="left top" at="left bottom">

    //content

</p:slideMenu>
```



Client Side API

Widget: *PrimeFaces.widget.TieredMenu*

Method	Params	Return Type	Description
show()	-	void	Shows overlay menu.
hide()	-	void	Hides overlay menu.
align()	-	void	Aligns overlay menu with trigger.

Skinning

TieredMenu resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-menu .ui-slidemenu	Container element of menu.
.ui-slidemenu-wrapper	Wrapper element for content.
.ui-slidemenu-content	Content container.
.ui-slidemenu-backward	Back navigator.
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.103 Slider

Slider is used to provide input with various customization options like orientation, display modes and skinning.

Info

Tag	**slider**
Component Class	**org.primefaces.component.slider.Slider**
Component Type	**org.primefaces.component.Slider**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SliderRenderer**
Renderer Class	**org.primefaces.component.slider.SliderRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
for	null	String	Id of the input text that the slider will be used for
display	null	String	Id of the component to display the slider value.
minValue	0	Integer	Minimum value of the slider
maxValue	100	Integer	Maximum value of the slider
style	null	String	Inline style of the container element
styleClass	null	String	Style class of the container element
animate	TRUE	Boolean	Boolean value to enable/disable the animated move when background of slider is clicked
type	horizontal	String	Sets the type of the slider, "horizontal" or "vertical".
step	1	Integer	Fixed pixel increments that the slider move in
disabled	FALSE	Boolean	Disables or enables the slider.

Name	Default	Type	Description
onSlideStart	null	String	Client side callback to execute when slide begins.
onSlide	null	String	Client side callback to execute during sliding.
onSlideEnd	null	String	Client side callback to execute when slide ends.

Getting started with Slider

Slider requires an input component to work with, *for* attribute is used to set the id of the input component whose input will be provided by the slider.

```
public class SliderBean {

	private int number;

	public int getNumber() {
		return number;
	}

	public void setNumber(int number) {
		this.number = number;
	}
}
```

```
<h:inputText id="number" value="#{sliderBean.number}" />

<p:slider for="number" />
```

Display Value

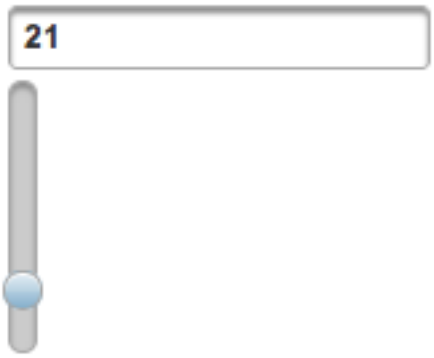
Using *display* feature, you can present a readonly display value and still use slider to provide input, in this case *for* should refer to a hidden input to bind the value.

```
<h:inputHidden id="number" value="#{sliderBean.number}" />
<h:outputText value="Set ratio to %" />
<h:outputText id="output" value="#{sliderBean.number}" />

<p:slider for="number" display="output" />
```

Vertical Slider

By default slider's orientation is horizontal, vertical sliding is also supported and can be set using the *type* attribute.

```
<h:inputText id="number" value="#{sliderController.number}" />

<p:slider for="number" type="vertical" minValue="0" maxValue="200"/>
```



Step

Step factor defines the interval between each point during sliding. Default value is one and it is customized using *step* option.

```
<h:inputText id="number" value="#{sliderBean.number}" />

<p:slider for="number" step="10" />
```

Animation

Sliding is animated by default, if you want to turn it of animate attribute set the *animate* option to false.

Boundaries

Maximum and minimum boundaries for the sliding is defined using minValue and maxValue attributes. Following slider can slide between -100 and +100.

```
<h:inputText id="number" value="#{sliderBean.number}" />

<p:slider for="number" minValue="-100" maxValue="100"/>
```

Client Side Callbacks

Slider provides three callbacks to hook-in your custom javascript, onSlideStart, onSlide and onSlideEnd. All of these callbacks receive two parameters; slide event and the ui object containing information about the event.

```
<h:inputText id="number" value="#{sliderBean.number}" />

<p:slider for="number" onSlideEnd="handleSlideEnd(event, ui)"/>
```

```
function handleSlideEnd(event, ui) {
      //ui.helper = Handle element of slider
      //ui.value = Current value of slider
}
```

Ajax Behavior Events

Slider provides one ajax behavior event called *slideEnd* that is fired when the slide completes. If you have a listener defined, it will be called by passing *org.primefaces.event.SlideEndEvent* instance. Example below adds a message and displays it using growl component when slide ends.

```
<h:inputText id="number" value="#{sliderBean.number}" />

<p:slider for="number">
      <p:ajax event="slideEnd" listener="#{sliderBean.onSlideEnd}" update="msgs" />
</p:slider>

<p:messages id="msgs" />
```

```
public class SliderBean {

      private int number;

      public int getNumber() {
            return number;
      }

      public void setNumber(int number) {
            this.number = number;
      }

      public void onSlideEnd(SlideEndEvent event) {
            int value = event.getValue();
            //add faces message
      }
}
```

Client Side API

Widget: *PrimeFaces.widget.Slider*

Method	Params	Return Type	Description
getValue()	-	Number	Returns the current value
setValue(value)	value: Value to set	void	Updates slider value with provided one.
disable()	index: Index of tab to disable	void	Disables slider.
enable()	index: Index of tab to enable	void	Enables slider.

Skinning

Slider resides in a main container which *style* and *styleClass* attributes apply. These attributes are handy to specify the dimensions of the slider.

Following is the list of structural style classes;

Class	Applies
.ui-slider	Main container element
.ui-slider-horizontal	Main container element of horizontal slider
.ui-slider-vertical	Main container element of vertical slider
.ui-slider-handle	Slider handle

As skinning style classes are global, see the main Skinning section for more information.

3.104 Spacer

Spacer is used to put spaces between elements.

Info

Tag	**spacer**
Component Class	**org.primefaces.component.spacer.Spacer**
Component Type	**org.primefaces.component.Spacer**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SpacerRenderer**
Renderer Class	**org.primefaces.component.spacer.SpacerRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
title	null	String	Advisory tooltip informaton.
style	null	String	Inline style of the spacer.
styleClass	null	String	Style class of the spacer.
width	null	String	Width of the space.
height	null	String	Height of the space.

Getting started with Spacer

Spacer is used by either specifying width or height of the space.

```
Spacer in this example separates this text <p:spacer width="100" height="10"> and
          <p:spacer width="100" height="10"> this text.
```

3.105 Spinner

Spinner is an input component to provide a numerical input via increment and decrement buttons.

Info

Tag	**spinner**
Component Class	**org.primefaces.component.spinner.Spinner**
Component Type	**org.primefaces.component.Spinner**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SpinnerRenderer**
Renderer Class	**org.primefaces.component.spinner.SpinnerRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
immediate	FALSE	Boolean	Boolean value that specifies the lifecycle phase the valueChangeEvents should be processed, when true the events will be fired at "apply request values", if immediate is set to false, valueChange Events are fired in "process validations" phase
required	FALSE	Boolean	Marks component as required
validator	null	Method Expr	A method binding expression that refers to a method validationg the input

Name	Default	Type	Description
valueChangeListener	null	Method Expr	A method binding expression that refers to a method for handling a valuchangeevent
requiredMessage	null	String	Message to be displayed when required field validation fails.
converterMessage	null	String	Message to be displayed when conversion fails.
validatorMessage	null	String	Message to be displayed when validation fields.
widgetVar	null	String	Name of the client side widget.
stepFactor	1	Double	Stepping factor for each increment and decrement
min	null	Double	Minimum boundary value
max	null	Double	Maximum boundary value
prefix	null	String	Prefix of the input
suffix	null	String	Suffix of the input
accesskey	null	String	Access key that when pressed transfers focus to the input element.
alt	null	String	Alternate textual description of the input field.
autocomplete	null	String	Controls browser autocomplete behavior.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables input field
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
maxlength	null	Integer	Maximum number of characters that may be entered in this field.
onblur	null	String	Client side callback to execute when input element loses focus.
onchange	null	String	Client side callback to execute when input element loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when input element is clicked.
ondblclick	null	String	Client side callback to execute when input element is double clicked.
onfocus	null	String	Client side callback to execute when input element receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over input element.

Name	Default	Type	Description
onkeypress	null	String	Client side callback to execute when a key is pressed and released over input element.
onkeyup	null	String	Client side callback to execute when a key is released over input element.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over input element
onmousemove	null	String	Client side callback to execute when a pointer button is moved within input element.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from input element.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto input element.
onmouseup	null	String	Client side callback to execute when a pointer button is released over input element.
onselect	null	String	Client side callback to execute when text within input element is selected by user.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
size	null	Integer	Number of characters used to determine the width of the input element.
style	null	String	Inline style of the input element.
styleClass	null	String	Style class of the input element.
tabindex	null	Integer	Position of the input element in the tabbing order.
title	null	String	Advisory tooltip informaton.

Getting Started with Spinner

Spinner is an input component and used just like a standard input text.

```
public class SpinnerBean {

      private int number;
      //getter and setter
}
```

```
<p:spinner value="#{spinnerBean.number}" />
```

Step Factor

Other than integers, spinner also support decimals so the fractional part can be controller with spinner as well. For decimals use the stepFactor attribute to specify stepping amount. Following example uses a stepFactor 0.25.

```
<p:spinner value="#{spinnerBean.number}" stepFactor="0.25"/>
```

```
public class SpinnerBean {

      private double number;

      //getter and setter

}
```

Output of this spinner would be;

After an increment happens a couple of times.

Prefix and Suffix

Prefix and Suffix options enable placing fixed strings on input field. Note that you would need to use a converter to avoid conversion errors since prefix/suffix will also be posted.

```
<p:spinner value="#{spinnerBean.number}" prefix="$" />
```

Boundaries

In order to restrict the boundary values, use *min* and *max* options.

```
<p:spinner value="#{spinnerBean.number}" min="0" max="100"/>
```

Ajax Spinner

Spinner can be ajaxified using client behaviors like f:ajax or p:ajax. In example below, an ajax request is done to update the outputtext with new value whenever a spinner button is clicked.

```
<p:spinner value="#{spinnerBean.number}">
     <p:ajax update="display" />
</p:spinner>

<h:outputText id="display" value="#{spinnerBean.number}" />
```

Skinning

Spinner resides in a container element that using *style* and *styleClass* applies.

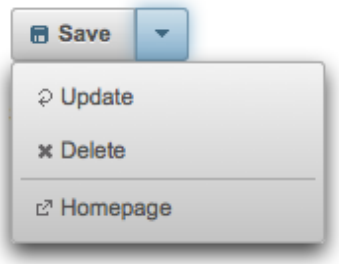
Following is the list of structural style classes;

Class	Applies
.ui-spinner	Main container element of spinner
.ui-spinner-input	Input field
.ui-spinner-button	Spinner buttons
.ui-spinner-button-up	Increment button
.ui-spinner-button-down	Decrement button

As skinning style classes are global, see the main Skinning section for more information.

3.106 SplitButton

SplitButton displays a command by default and additional ones in an overlay.



Info

Tag	**splitButton**
Component Class	**org.primefaces.component.splitbutton.SplitButton**
Component Type	**org.primefaces.component.SplitButton**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SplitButtonRenderer**
Renderer Class	**org.primefaces.component.splitbutton.SplitButtonRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	String	Label for the button
action	null	MethodExpr/ String	A method expression or a String outcome that'd be processed when button is clicked.
actionListener	null	MethodExpr	An actionlistener that'd be processed when button is clicked.
immediate	FALSE	Boolean	Boolean value that determines the phaseId, when true actions are processed at apply_request_values, when false at invoke_application phase.
type	submit	String	Sets the behavior of the button.
ajax	TRUE	Boolean	Specifies the submit mode, when set to true(default), submit would be made with Ajax.

Name	Default	Type	Description
async	FALSE	Boolean	When set to true, ajax requests are not queued.
process	null	String	Component(s) to process partially instead of whole view.
update	null	String	Component(s) to be updated with ajax.
onstart	null	String	Client side callback to execute before ajax request is begins.
oncomplete	null	String	Client side callback to execute when ajax request is completed.
onsuccess	null	String	Client side callback to execute when ajax request succeeds.
onerror	null	String	Client side callback to execute when ajax request fails.
global	TRUE	Boolean	Defines whether to trigger ajaxStatus or not.
partialSubmit	TRUE	Boolean	Enables serialization of values belonging to the partially processed components only.
style	null	String	Inline style of the button element.
styleClass	null	String	StyleClass of the button element.
onblur	null	String	Client side callback to execute when button loses focus.
onchange	null	String	Client side callback to execute when button loses focus and its value has been modified since gaining focus.
onclick	null	String	Client side callback to execute when button is clicked.
ondblclick	null	String	Client side callback to execute when button is double clicked.
onfocus	null	String	Client side callback to execute when button receives focus.
onkeydown	null	String	Client side callback to execute when a key is pressed down over button.
onkeypress	null	String	Client side callback to execute when a key is pressed and released over button.
onkeyup	null	String	Client side callback to execute when a key is released over button.
onmousedown	null	String	Client side callback to execute when a pointer button is pressed down over button.
onmousemove	null	String	Client side callback to execute when a pointer button is moved within button.
onmouseout	null	String	Client side callback to execute when a pointer button is moved away from button.
onmouseover	null	String	Client side callback to execute when a pointer button is moved onto button.

Name	Default	Type	Description
onmouseup	null	String	Client side callback to execute when a pointer button is released over button.
onselect	null	String	Client side callback to execute when text within button is selected by user.
accesskey	null	String	Access key that when pressed transfers focus to the button.
alt	null	String	Alternate textual description of the button.
dir	null	String	Direction indication for text that does not inherit directionality. Valid values are LTR and RTL.
disabled	FALSE	Boolean	Disables the button.
image	null	String	Style class for the button icon. (deprecated: use icon)
label	null	String	A localized user presentable name.
lang	null	String	Code describing the language used in the generated markup for this component.
tabindex	null	Integer	Position of the button element in the tabbing order.
title	null	String	Advisory tooltip information.
readonly	FALSE	Boolean	Flag indicating that this component will prevent changes by the user.
icon	null	String	Icon of the button as a css class.
iconPos	left	String	Position of the icon.
widgetVar	null	String	Name of the client side widget.

Getting started with SplitButton

SplitButton usage is similar to a regular commandButton. Additional commands are placed inside the component and displayed in an overlay. In example below, clicking the save button invokes save method of the bean and updates messages. Nested options defined as menuitems do ajax, non-ajax requests as well as regular navigation to an external url.

```
<p:splitButton value="Save" actionListener="#{buttonBean.save}" update="messages"
      icon="ui-icon-disk">
      <p:menuitem value="Update" actionListener="#{buttonBean.update}"
                  update="messages" icon="ui-icon-arrowrefresh-1-w"/>
      <p:menuitem value="Delete" actionListener="#{buttonBean.delete}" ajax="false"
                  icon="ui-icon-close"/>
      <p:separator />
      <p:menuitem value="Homepage" url="http://www.primefaces.org"
                  icon="ui-icon-extlink"/>
</p:splitButton>
```

Client Side API

Widget: *PrimeFaces.widget.SplitButton*

Method	Params	Return Type	Description
show()	-	void	Displays overlay.
hide()	-	void	Hides overlay.

Skinning

SplitButton renders a container element which *style* and *styleClass* applies.

Following is the list of structural style classes;

Style Class	Applies
.ui-splitbutton	Container element.
.ui-button	Button element
.ui-splitbutton-menubutton	Dropdown button
.ui-button-text-only	Button element when icon is not used
.ui-button-text	Label of button
.ui-menu	Container element of menu
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.107 Submenu

Submenu is nested in menu components and represents a sub menu items.

Info

Tag	**submenu**
Component Class	**org.primefaces.component.submenu.Submenu**
Component Type	**org.primefaces.component.Submenu**
Component Family	**org.primefaces.component**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
label	null	String	Label of the submenu header.
icon	null	String	Icon of a submenu, see menuitem to see how it is used
style	null	String	Inline style of the submenu.
styleClass	null	String	Style class of the submenu.

Getting started with Submenu

Please see Menu or Menubar section to find out how submenu is used with the menus.

3.108 Stack

Stack is a navigation component that mimics the stacks feature in Mac OS X.

Info

Tag	**stack**
Component Class	**org.primefaces.component.stack.Stack**
Component Type	**org.primefaces.component.Stack**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.StackRenderer**
Renderer Class	**org.primefaces.component.stack.StackRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
icon	null	String	An optional image to contain stacked items.
openSpeed	300	String	Speed of the animation when opening the stack.
closeSpeed	300	Integer	Speed of the animation when opening the stack.
widgetVar	null	String	Name of the client side widget.
model	null	MenuModel	MenuModel instance to create menus programmatically
expanded	FALSE	Boolean	Whether to display stack as expanded or not.

Getting started with Stack

Each item in the stack is represented with menuitems. Stack below has five items with different icons and labels.

```
<p:stack icon="/images/stack/stack.png">
  <p:menuitem value="Aperture" icon="/images/stack/aperture.png" url="#"/>
  <p:menuitem value="Photoshop" icon="/images/stack/photoshop.png" url="#"/>
  //...
</p:stack>
```

Initially stack will be rendered in collapsed mode;

Location

Stack is a fixed positioned element and location can be change via css. There's one important css selector for stack called *.ui-stack*. Override this style to change the location of stack.

```
.ui-stack {
      bottom: 28px;
      right: 40px;
}
```

Dynamic Menus

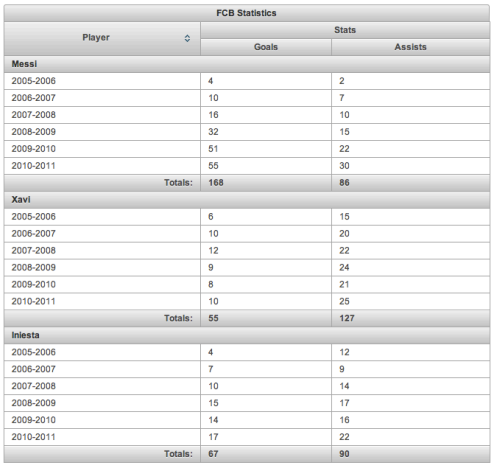
Menus can be created programmatically as well, see the dynamic menus part in menu component section for more information and an example.

Skinning

Class	Applies
.ui-stack	Main container element of stack
.ui-stack ul li	Each item in stack
.ui-stack ul li img	Icon of a stack item
.ui-stack ul li span	Label of a stack item

3.109 SubTable

SummaryRow is a helper component of datatable used for grouping.



FCB Statistics		
Player	Stats	
	Goals	Assists
Messi		
2005-2006	4	2
2006-2007	10	7
2007-2008	16	10
2008-2009	32	15
2009-2010	51	22
2010-2011	55	30
Totals:	168	86
Xavi		
2005-2006	6	15
2006-2007	10	20
2007-2008	12	22
2008-2009	9	24
2009-2010	8	21
2010-2011	10	25
Totals:	55	127
Iniesta		
2005-2006	4	12
2006-2007	7	9
2007-2008	10	14
2008-2009	15	17
2009-2010	14	16
2010-2011	17	22
Totals:	67	90

Info

Tag	**subTable**
Component Class	**org.primefaces.component.subtable.SubTable**
Component Type	**org.primefaces.component.SubTable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SubTableRenderer**
Renderer Class	**org.primefaces.component.subtable.SubTableRenderer**

Attributes

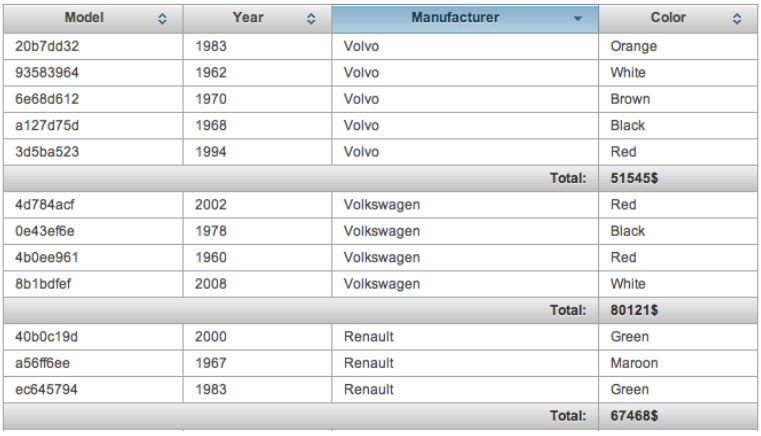
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Data of the component.
var	null	String	Name of the data iterator.

Getting started with SubTable

See DataTable section for more information.

3.110 SummaryRow

SummaryRow is a helper component of datatable used for dynamic grouping.



Model	Year	Manufacturer	Color
20b7dd32	1983	Volvo	Orange
93583964	1962	Volvo	White
6e68d612	1970	Volvo	Brown
a127d75d	1968	Volvo	Black
3d5ba523	1994	Volvo	Red
		Total:	51545$
4d784acf	2002	Volkswagen	Red
0e43ef6e	1978	Volkswagen	Black
4b0ee961	1960	Volkswagen	Red
8b1bdfef	2008	Volkswagen	White
		Total:	80121$
40b0c19d	2000	Renault	Green
a56ff6ee	1967	Renault	Maroon
ec645794	1983	Renault	Green
		Total:	67468$

Info

Tag	**summaryRow**
Component Class	**org.primefaces.component.summaryrow.SummaryRow**
Component Type	**org.primefaces.component.SummaryRow**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.SummaryRowRenderer**
Renderer Class	**org.primefaces.component.summaryrow.SummaryRowRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
listener	null	MethodExpr	Method expression to execute before rendering summary row. (e.g. to calculate totals).

Getting started with SummaryRow

See DataTable section for more information.

3.111 Tab

Tab is a generic container component used by other PrimeFaces components such as tabView and accordionPanel.

Info

Tag	**tab**
Component Class	**org.primefaces.component.TabView.Tab**
Component Type	**org.primefaces.component.Tab**
Component Family	**org.primefaces.component**

Attributes

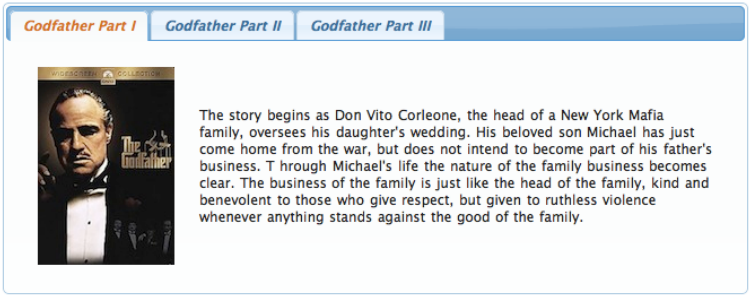
Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
title	null	Boolean	Title text of the tab
titleStyle	null	String	Inline style of the tab.
titleStyleClass	null	String	Style class of the tab.
disabled	FALSE	Boolean	Disables tab element.
closable	FALSE	Boolean	Makes the tab closable when enabled.
titletip	null	String	Tooltip of the tab header.

Getting started with the Tab

See the sections of components who utilize tab component for more information. As tab is a shared component, not all attributes may apply to the components that use tab.

3.112 TabView

TabView is a tabbed panel component featuring client side tabs, dynamic content loading with ajax and content transition effects.


Godfather Part I
Godfather Part II
Godfather Part III
The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. T hrough Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.

Info

Tag	**tabView**
Component Class	**org.primefaces.component. tabview.TabView**
Component Type	**org.primefaces.component.TabView**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TabViewRenderer**
Renderer Class	**org.primefaces.component.tabview.TabViewRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean.
widgetVar	null	String	Variable name of the client side widget.
activeIndex	0	Integer	Index of the active tab.
effect	null	String	Name of the transition effect.
effectDuration	null	String	Duration of the transition effect.
dynamic	FALSE	Boolean	Enables lazy loading of inactive tabs.

Name	Default	Type	Description
cache	TRUE	Boolean	When tab contents are lazy loaded by ajax toggleMode, caching only retrieves the tab contents once and subsequent toggles of a cached tab does not communicate with server. If caching is turned off, tab contents are reloaded from server each time tab is clicked.
onTabChange	null	String	Client side callback to execute when a tab is clicked.
onTabShow	null	String	Client side callback to execute when a tab is shown.
style	null	String	Inline style of the main container.
styleClass	null	String	Style class of the main container.
var	null	String	Name of iterator to refer an item in collection.
value	null	Object	Collection model to display dynamic tabs.
orientation	top	String	Orientation of tab headers.

Getting started with the TabView

TabView requires one more child tab components to display. Titles can also be defined by using "title" facet.

```
<p:tabView>
      <p:tab title="Tab One">
            <h:outputText value="Lorem" />
      </p:tab>
      <p:tab title="Tab Two">
            <h:outputText value="Ipsum" />
      </p:tab>
      <p:tab title="Tab Three">
            <h:outputText value="Dolor" />
      </p:tab>
</p:tabView>
```

Dynamic Tabs

There're two toggleModes in tabview, *non-dynamic* (default) and *dynamic*. By default, all tab contents are rendered to the client, on the other hand in dynamic mode, only the active tab contents are rendered and when an inactive tab header is selected, content is loaded with ajax. Dynamic mode is handy in reducing page size, since inactive tabs are lazy loaded, pages will load faster. To enable dynamic loading, simply set *dynamic* option to true.

```
<p:tabView dynamic="true">
      //tabs
</p:tabView>
```

Content Caching

Dynamically loaded tabs cache their contents by default, by doing so, reactivating a tab doesn't result in an ajax request since contents are cached. If you want to reload content of a tab each time a tab is selected, turn off caching by *cache* to false.

Effects

Content transition effects are controlled with *effect* and *effectDuration* attributes. EffectDuration specifies the speed of the effect, *slow*, *normal* (default) and *fast* are the valid options.

```
<p:tabView effect="fade" effectDuration="fast">
      //tabs
</p:tabView>
```

Ajax Behavior Events

tabChange and *tabClose* are the ajax behavior events of tabview that are executed when a tab is changed and closed respectively. Here is an example of a tabChange behavior implementation;

```
<p:tabView>
      <p:ajax event="tabChange" listener="#{bean.onChange}" />

      //tabs
</p:tabView>
```

```
public void onChange(TabChangeEvent event) {
      //Tab activeTab = event.getTab();
      //...
 }
```

Your listener(if defined) will be invoked with an *org.primefaces.event.TabChangeEvent* instance that contains a reference to the new active tab and the accordion panel itself. For tabClose event, listener will be passed an instance of *org.primefaces.event.TabCloseEvent.*

Dynamic Number of Tabs

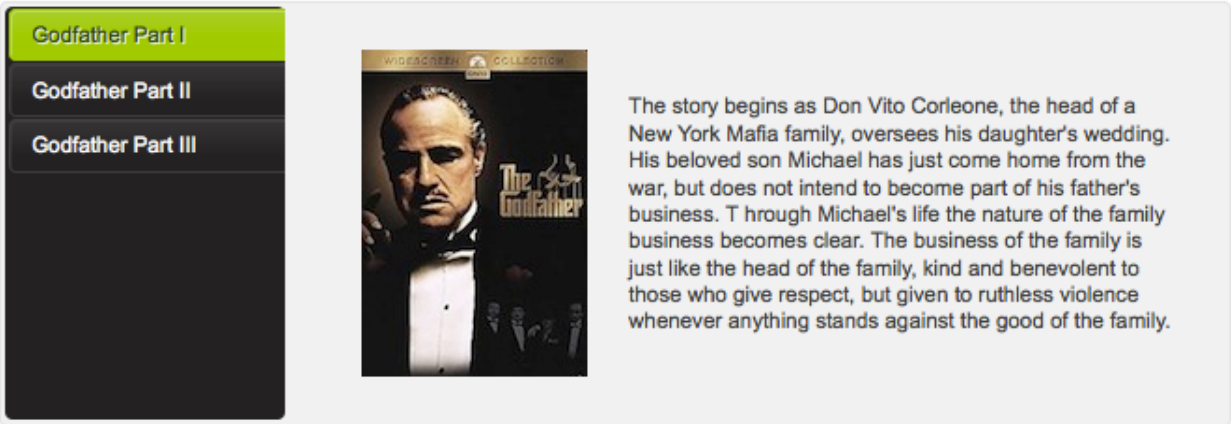
When the tabs to display are not static, use the built-in iteration feature similar to ui:repeat.

```
<p:tabView value="#{bean.list}" var="listItem">
      <p:tab title="#{listItem.propertyA}">
            <h:outputText value= "#{listItem.propertyB}"/>
            ...More content
      </p:tab>
</p:tabView>
```

Orientations

Tabview supports four different orientations mode, *top(default)*, *left*, *right* and *bottom*.

```
<p:tabView orientation="left">
      //tabs
 </p:tabView>
```


Godfather Part I
Godfather Part II
Godfather Part III
The Godfather
The story begins as Don Vito Corleone, the head of a New York Mafia family, oversees his daughter's wedding. His beloved son Michael has just come home from the war, but does not intend to become part of his father's business. T hrough Michael's life the nature of the family business becomes clear. The business of the family is just like the head of the family, kind and benevolent to those who give respect, but given to ruthless violence whenever anything stands against the good of the family.

Client Side Callbacks

Tabview has two callbacks for client side. *onTabChange* is executed when an inactive tab is clicked and *onTabShow* is executed when an inactive tab becomes active to be shown.

```
<p:tabView onTabChange="handleTabChange(event, index)">
      //tabs
</p:tabView>

function handleTabChange(index) {
      //index = Index of the new tab
}
```

Client Side API

Widget: *PrimeFaces.widget.TabView.*

Method	Params	Return Type	Description
select(index)	index: Index of tab to display	void	Activates tab with given index
selectTab(index)	index: Index of tab to display	void	(Deprecated, use select instead) Activates tab with given index
disable(index)	index: Index of tab to disable	void	Disables tab with given index
enable(index)	index: Index of tab to enable	void	Enables tab with given index
remove(index)	index: Index of tab to remove	void	Removes tab with given index
getLength()	-	Number	Returns the number of tabs
getActiveIndex()	-	Number	Returns index of current tab

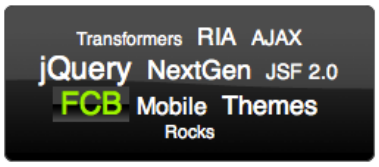
Skinning

As skinning style classes are global, see the main Skinning section for more information. Following is the list of structural style classes.

Class	Applies
.ui-tabs	Main tabview container element
.ui-tabs-nav	Main container of tab headers
.ui-tabs-panel	Each tab container

3.113 TagCloud

TagCloud displays a collection of tag with different strengths.



Info

Tag	**tagCloud**
Component Class	**org.primefaces.component.tagcloud.TagCloud**
Component Type	**org.primefaces.component.TagCloud**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TagCloudRenderer**
Renderer Class	**org.primefaces.component.tagcloud.TagCloudRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
model	null	TagCloudModel	Backing tag cloud model.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.

Getting started with the TagCloud

TagCloud requires a backend TagCloud model to display.

```
<p:tagCloud model="#{timelineBean.model}" />
```

```
public class TagCloudBean {

    private TagCloudModel model;

    public TagCloudBean() {
        model = new DefaultTagCloudModel();
        model.addTag(new DefaultTagCloudItem("Transformers", "#", 1));
        //more
    }

    //getter
}
```

TagCloud API

org.primefaces.model.tagcloud.TagCloudModel

Method	Description
List<TagCLoudItem> getTags()	Returns all tags in model.
void addTag(TagCloudItem item)	Adds a tag.
void removeTag(TagCloudItem item)	Removes a tag.
void clear()	Removes all tags.

PrimeFaces provides *org.primefaces.model.tagcloud.DefaultTagCloudModel* as the default implementation.

---

org.primefaces.model.tagcloud.TagCloudItem

Method	Description
String getLabel()	Returns label of the tag.
String getUrl()	Returns url of the tag.
int getStrength()	Returns strength of the tag between 1 and 5.

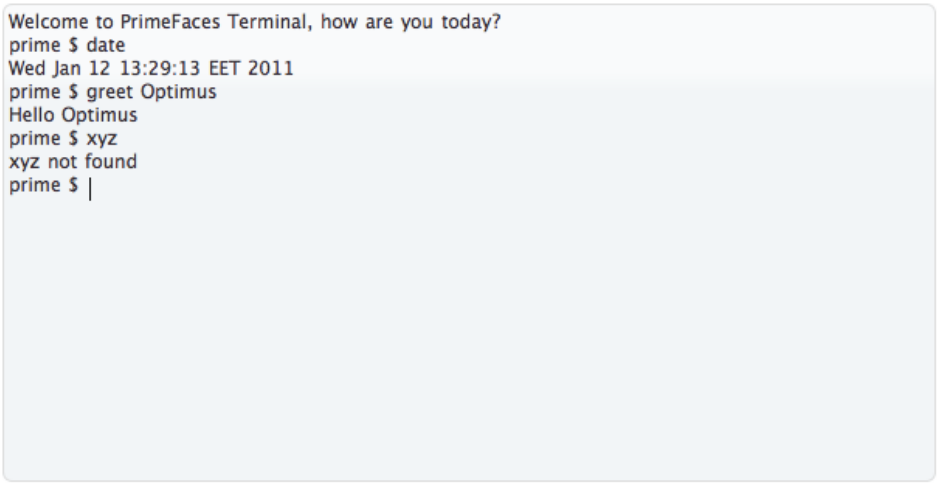
PrimeFaces provides *org.primefaces.model.tagcloud.DefaultTagCloudItem* as the default implementation.

Skinning

TagCloud resides in a container element that *style* and *styleClass* attributes apply. *.ui-tagcloud* applies to main container and *.ui-tagcloud-strength-[1,5]* applies to each tag. As skinning style classes are global, see the main Skinning section for more information.

3.114 Terminal

Terminal is an ajax powered web based terminal that brings desktop terminals to JSF.



Info

Tag	**terminal**
Component Class	**org.primefaces.component.terminal.Terminal**
Component Type	**org.primefaces.component.Terminal**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TerminalRenderer**
Renderer Class	**org.primefaces.component.terminal.TerminalRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
width	null	String	Width of the terminal
height	null	String	Height of the terminal
welcomeMessage	null	String	Welcome message to be displayed on initial load.
prompt	prime $	String	Primary prompt text.

Name	Default	Type	Description
commandHandler	null	MethodExpr	Method to be called with arguments to process.
widgetVar	null	String	Name of the client side widget.

Getting started with the Terminal

A command handler is necessary to interpret commands entered in terminal.

```
<p:terminal commandHandler="#{terminalBean.handleCommand}" />
```

```
public class TerminalBean {

      public String handleCommand(String command, String[] params) {
            if(command.equals("greet"))
                  return "Hello " + params[0];
            else if(command.equals("date"))
                  return new Date().toString();
            else
                  return command + " not found";
      }
}
```

Whenever a command is sent to the server, handleCommand method is invoked with the command name and the command arguments as a String array.

Focus

To add focus on terminal, use client side api, following example shows how to add focus on a terminal nested inside a dialog;

```
<p:commandButton type="Show Terminal" type="button"
            onclick="dlg.show();term.focus();"/>

     <p:dialog widgetVar="dlg" width="600" height="400" header="Terminal">
       <p:terminal widgetVar="term"
                  commandHandler="#{terminalBean.handleCommand}" width="590px" />
</p:dialog>
```

3.115 ThemeSwitcher

ThemeSwitcher enables switching PrimeFaces themes on the fly with no page refresh.

Info

Tag	**themeSwitcher**
Component Class	**org.primefaces.component.themeswitcher.ThemeSwitcher**
Component Type	**org.primefaces.component.ThemeSwitcher**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ThemeSwitcherRenderer**
Renderer Class	**org.primefaces.component.themeswitcher.ThemeSwitcherRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
effect	blind	String	Name of the animation.
effectDuration	400	Integer	Duration of the animation in milliseconds.
disabled	FALSE	Boolean	Disables the component.
label	null	String	User presentable name.
onchange	null	String	Client side callback to execute on theme change.
style	null	String	Inline style of the component.

Name	Default	Type	Description
styleClass	null	String	Style class of the component.
var	null	String	Variable name to refer to each item.
height	null	Integer	Height of the panel.
tabindex	null	Integer	Position of the element in the tabbing order.

Getting Started with the ThemeSwitcher

ThemeSwitcher usage is very similar to selectOneMenu.

```
<p:themeSwitcher style="width:150px">
      <f:selectItem itemLabel="Choose Theme" itemValue="" />
      <f:selectItems value="#{bean.themes}" />
</p:themeSwitcher>
```

Stateful ThemeSwitcher

By default, themeswitcher just changes the theme on the fly with no page refresh, in case you'd like to get notified when a user changes the theme (e.g. to update user preferences), you can use an ajax behavior.

```
<p:themeSwitcher value="#{bean.theme}" effect="fade">
      <f:selectItem itemLabel="Choose Theme" itemValue="" />
      <f:selectItems value="#{themeSwitcherBean.themes}" />
      <p:ajax listener="#{bean.saveTheme}" />
</p:themeSwitcher>
```

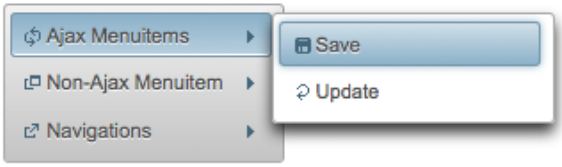
Advanced ThemeSwitcher

ThemeSwitcher supports displaying custom content so that you can show theme previews.

```
<p:themeSwitcher>
      <f:selectItem itemLabel="Choose Theme" itemValue="" />
      <f:selectItems value="#{themeSwitcherBean.advancedThemes}" var="theme"
            itemLabel="#{theme.name}" itemValue="#{theme}"/>

      <p:column>
            <p:graphicImage value="/images/themes/#{t.image}"/>
      </p:column>

      <p:column>
            #{t.name}
      </p:column>
</p:themeSwitcher>
```

3.116 TieredMenu

TieredMenu is used to display nested submenus with overlays.



Info

Tag	**tieredMenu**
Component Class	**org.primefaces.component.tieredmenu.TieredMenu**
Component Type	**org.primefaces.component.TieredMenu**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TieredMenuRenderer**
Renderer Class	**org.primefaces.component.tieredmenu.TieredMenuRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
model	null	org.primefaces.model.MenuModel	MenuModel instance for programmatic menu.
style	null	String	Inline style of the component.
styleClass	null	String	Style class of the component.
autoDisplay	TRUE	Boolean	Defines whether the first level of submenus will be displayed on mouseover or not. When set to false, click event is required to display.
trigger	null	String	Id of the component whose triggerEvent will show the dynamic positioned menu.
my	null	String	Corner of menu to align with trigger element.

Name	Default	Type	Description
at	null	String	Corner of trigger to align with menu element.
overlay	FALSE	Boolean	Defines positioning, when enabled menu is displayed with absolute position relative to the trigger. Default is false, meaning static positioning.
triggerEvent	click	String	Event name of trigger that will show the dynamic positioned menu.

Getting started with the TieredMenu

TieredMenu consists of submenus and menuitems, submenus can be nested and each nested submenu will be displayed in an overlay.

```
<p:tieredMenu>
   <p:submenu label="Ajax Menuitems" icon="ui-icon-refresh">
       <p:menuitem value="Save" actionListener="#{buttonBean.save}"
                           update="messages" icon="ui-icon-disk" />
      <p:menuitem value="Update" actionListener="#{buttonBean.update}"
                           update="messages" icon="ui-icon-arrowrefresh-1-w" />
      </p:submenu>

      <p:submenu label="Non-Ajax Menuitem" icon="ui-icon-newwin">
            <p:menuitem value="Delete" actionListener="#{buttonBean.delete}"
                  update="messages" ajax="false" icon="ui-icon-close"/>
      </p:submenu>

      <p:separator />

      <p:submenu label="Navigations" icon="ui-icon-extlink">
            <p:submenu label="Prime Links">
                  <p:menuitem value="Prime" url="http://www.prime.com.tr" />
                  <p:menuitem value="PrimeFaces" url="http://www.primefaces.org" />
            </p:submenu>
            <p:menuitem value="Mobile" url="/mobile" />
      </p:submenu>
</p:tieredMenu>
```

AutoDisplay

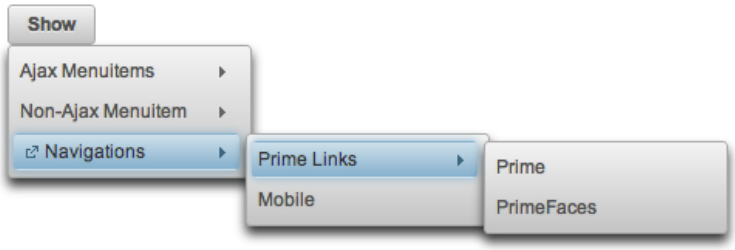
By default, submenus are displayed when mouse is over root menuitems, set autoDisplay to false to require a click on root menuitems to enable autoDisplay mode.

```
<p:tieredMenu autoDisplay="false">

   //content

</p:tieredMenu>
```

Overlay

TieredMenu can be positioned relative to a trigger component, following sample attaches a tieredMenu to the button so that whenever the button is clicked tieredMenu will be displayed in an overlay itself.

```
 <p:commandButton type="button" value="Show" id="btn" />

<p:tieredMenu autoDisplay="false" trigger="btn" my="left top" at="left bottom">

    //content

</p:tieredMenu>
```



Client Side API

Widget: *PrimeFaces.widget.TieredMenu*

Method	Params	Return Type	Description
show()	-	void	Shows overlay menu.
hide()	-	void	Hides overlay menu.
align()	-	void	Aligns overlay menu with trigger.

Skinning

TieredMenu resides in a main container which *style* and *styleClass* attributes apply. Following is the list of structural style classes;

Style Class	Applies
.ui-menu .ui-tieredmenu	Container element of menu.
.ui-menu-list	List container
.ui-menuitem	Each menu item
.ui-menuitem-link	Anchor element in a link item
.ui-menuitem-text	Text element in an item

As skinning style classes are global, see the main Skinning section for more information.

3.117 Toolbar

Toolbar is a horizontal grouping component for commands and other content.

Info

Tag	**toolbar**
Component Class	**org.primefaces.component.toolbar.Toolbar**
Component Type	**org.primefaces.component.Toolbar**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.ToolbarRenderer**
Renderer Class	**org.primefaces.component.toolbar.ToolbarRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.

Getting Started with the Toolbar

Toolbar has two placeholders(left and right) that are defined with toolbarGroup component.

```
<p:toolbar>
      <p:toolbarGroup align="left">
      </p:toolbarGroup>

      <p:toolbarGroup align="right">
      </p:toolbarGroup>
 </p:toolbar>
```

Any number of components can be placed inside toolbarGroups. Additionally p:separator component can be used to separate items in toolbar. Here is an example;

```
<p:toolbar>
      <p:toolbarGroup align="left">
            <p:commandButton type="push" value="New" image="ui-icon-document" />
            <p:commandButton type="push" value="Open" image="ui-icon-folder-open"/>

            <p:separator />

            <p:commandButton type="push" title="Save" image="ui-icon-disk"/>
            <p:commandButton type="push" title="Delete" image="ui-icon-trash"/>
            <p:commandButton type="push" title="Print" image="ui-icon-print"/>
      </p:toolbarGroup>

      <p:divider />

      <p:toolbarGroup align="right">
            <p:menuButton value="Navigate">
                  <p:menuitem value="Home" url="#" />
                  <p:menuitem value="Logout" url="#" />
            </p:menuButton>
      </p:toolbarGroup>
</p:toolbar>
```

Skinning

Toolbar resides in a container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-toolbar	Main container
.ui-toolbar .ui-separator	Divider in a toolbar
.ui-toolbar-group-left	Left toolbarGroup container
.ui-toolbar-group-right	Right toolbarGroup container

As skinning style classes are global, see the main Skinning section for more information.

3.118 ToolbarGroup

ToolbarbarGroup is a helper component for Toolbar component to define placeholders.

Info

Tag	**toolbarGroup**
Component Class	**org.primefaces.component.toolbar.ToolbarGroup**
Component Type	**org.primefaces.component.ToolbarGroup**
Component Family	**org.primefaces.component**

Attributes

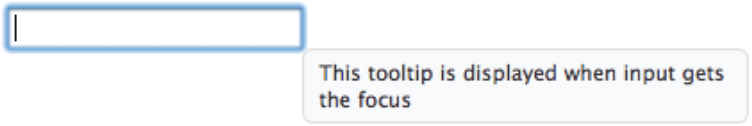
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
align	null	String	Defines the alignment within toolbar, valid values are *left* and *right*.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.

Getting Started with the ToolbarGroup

See toolbar documentation for more information about how Toolbar Group is used.

3.119 Tooltip

Tooltip goes beyond the legacy html title attribute by providing custom effects, events, html content and advance theme support.



Info

Tag	**tooltip**
Component Class	**org.primefaces.component.tooltip.Tooltip**
Component Type	**org.primefaces.component.Tooltip**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TooltipRenderer**
Renderer Class	**org.primefaces.component.tooltip.TooltipRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	Value of the component than can be either an EL expression of a literal text
converter	null	Converter/ String	An el expression or a literal text that defines a converter for the component. When it's an EL expression, it's resolved to a converter instance. In case it's a static text, it must refer to a converter id
widgetVar	null	String	Name of the client side widget.
showEvent	mouseover	String	Event displaying the tooltip.
showEffect	fade	String	Effect to be used for displaying.
hideEvent	mouseout	String	Event hiding the tooltip.
hideEffect	fade	String	Effect to be used for hiding.

Name	Default	Type	Description
for	null	String	Id of the component to attach the tooltip.
style	null	String	Inline style of the tooltip.
styleClass	null	String	Style class of the tooltip.

Getting started with the Tooltip

Tooltip is used by attaching it to a target component. Tooltip value can also be retrieved from target's title, so following is same;

```
<h:inputSecret id="pwd" value="#{myBean.password}" />
<p:tooltip for="pwd" value="Only numbers"/>
```

```
<h:inputSecret id="pwd" value="#{myBean.password}" title="Only numbers"/>
<p:tooltip for="pwd"/>
```

Events and Effects

A tooltip is shown on mouseover event and hidden when mouse is out by default. If you need to change this behaviour use the showEvent and hideEvent feature. Tooltip below is displayed when the input gets the focus and hidden with onblur.

```
<h:inputSecret id="pwd" value="#{myBean.password}" />
<p:tooltip for="pwd" value="Password must contain only numbers"
     showEvent="focus" hideEvent="blur" showEffect="blind" hideEffect="explode" />
```

Available options for effects are;

- blind
- bounce
- clip
- drop
- explode
- fold
- highlight
- puff
- pulsate
- scale
- shake
- size
- slide

Html Content

Another powerful feature of tooltip is the ability to display custom content as a tooltip not just plain texts. An example is as follows;

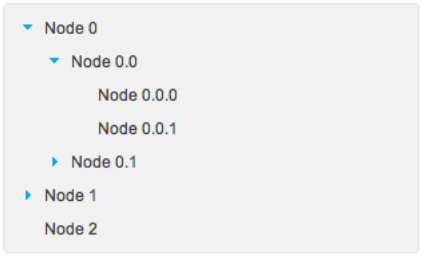
```
 <h:outputLink id="lnk" value="#">
      <h:outputText value="PrimeFaces Home" />
</h:outputLink>

<p:tooltip for="lnk">
      <p:graphicImage value="/images/prime_logo.png" />
      <h:outputText value="Visit PrimeFaces Home" />
</p:tooltip>
```

Skinning

Tooltip has only *.ui-tooltip* as a style class and is styled with global skinning selectors, see main skinning section for more information.

3.120 Tree

Tree is is used for displaying hierarchical data and creating site navigations.



Info

Tag	**tree**
Component Class	**org.primefaces.component.tree.Tree**
Component Type	**org.primefaces.component.Tree**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TreeRenderer**
Renderer Class	**org.primefaces.component.tree.TreeRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
widgetVar	null	String	Name of the client side widget.
value	null	Object	A TreeNode instance as the backing model.
var	null	String	Name of the request-scoped variable that'll be used to refer each treenode data.
dynamic	FALSE	Boolean	Specifies the ajax/client toggleMode
cache	TRUE	Boolean	Specifies caching on dynamically loaded nodes. When set to true expanded nodes will be kept in memory.

Name	Default	Type	Description
onNodeClick	null	String	Javascript event to process when a tree node is clicked.
selection	null	Object	TreeNode array to reference the selections.
style	null	String	Style of the main container element of tree
styleClass	null	String	Style class of the main container element of tree
selectionMode	null	String	Defines the selectionMode
highlight	TRUE	Boolean	Highlights nodes on hover when selection is enabled.
datakey	null	Object	Unique key of the data presented by nodes.

Getting started with the Tree

Tree is populated with a *org.primefaces.model.TreeNode* instance which corresponds to the root. TreeNode API has a hierarchical data structure and represents the data to be populated in tree.

```
public class TreeBean {

      private TreeNode root;

      public TreeBean() {
            root = new TreeNode("Root", null);
            TreeNode node0 = new TreeNode("Node 0", root);
            TreeNode node1 = new TreeNode("Node 1", root);
            TreeNode node2 = new TreeNode("Node 2", root);

            TreeNode node00 = new TreeNode("Node 0.0", node0);
            TreeNode node01 = new TreeNode("Node 0.1", node0);

            TreeNode node10 = new TreeNode("Node 1.0", node1);
            TreeNode node11 = new TreeNode("Node 1.1", node1);

            TreeNode node000 = new TreeNode("Node 0.0.0", node00);
            TreeNode node001 = new TreeNode("Node 0.0.1", node00);
            TreeNode node010 = new TreeNode("Node 0.1.0", node01);

            TreeNode node100 = new TreeNode("Node 1.0.0", node10);
      }

      //getter
}
```

Then specify a UI treeNode component as a child to display the nodes.

```
<p:tree value="#{treeBean.root}" var="node">
      <p:treeNode>
            <h:outputText value="#{node}"/>
      </p:treeNode>
</p:tree>
```

TreeNode vs p:TreeNode

TreeNode API is used to create the node model and consists of *org.primefaces.model.TreeNode* instances, on the other hand <p:treeNode /> tag represents a component of type *org.primefaces.component.tree.UITreeNode*. You can bind a TreeNode to a particular p:treeNode using the *type* name. Document Tree example in upcoming section demonstrates a sample usage.

TreeNode API

TreeNode has a simple API to use when building the backing model. For example if you call node.setExpanded(true) on a particular node, tree will render that node as expanded.

Property	Type	Description
type	String	type of the treeNode name, default type name is "default".
data	Object	Encapsulated data
children	List<TreeNode>	List of child nodes
parent	TreeNode	Parent node
expanded	Boolean	Flag indicating whether the node is expanded or not

Dynamic Tree

Tree is non-dynamic by default and toggling happens on client-side. In order to enable ajax toggling set dynamic setting to true.

```
<p:tree value="#{treeBean.root}" var="node" dynamic="true">
      <p:treeNode>
            <h:outputText value="#{node}"/>
      </p:treeNode>
</p:tree>
```

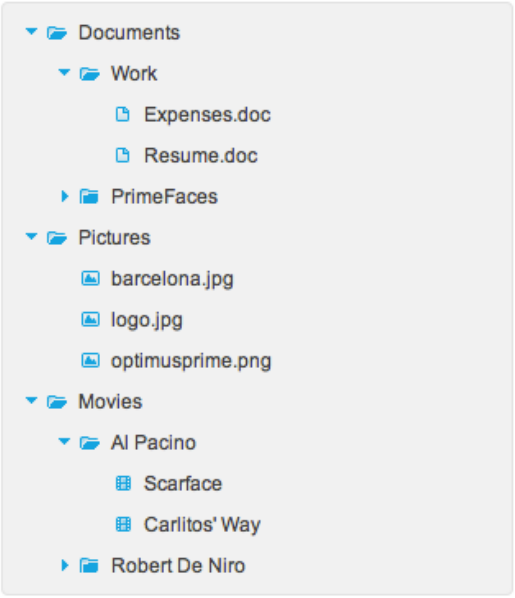
Non-Dynamic: When toggling is set to client all the treenodes in model are rendered to the client and tree is created, this mode is suitable for relatively small datasets and provides fast user interaction. On the otherhand it's not suitable for large data since all the data is sent to the client.

Dynamic: Dynamic mode uses ajax to fetch the treenodes from server side on demand, compared to the client toggling, dynamic mode has the advantage of dealing with large data because only the child nodes of the root node is sent to the client initially and whole tree is lazily populated. When a

node is expanded, tree only loads the children of the particular expanded node and send to the client for display.

Multiple TreeNode Types

It's a common requirement to display different TreeNode types with a different UI (eg icon). Suppose you're using tree to visualize a company with different departments and different employees, or a document tree with various folders, files each having a different formats (music, video). In order to solve this, you can place more than one <p:treeNode /> components each having a different type and use that "type" to bind TreeNode's in your model. Following example demonstrates a document explorer. To begin with here is the final output;



Document Explorer is implemented with four different <p:treeNode /> components and additional CSS skinning to visualize expanded/closed folder icons.

```
<p:tree value="#{bean.root}" var="doc">
      <p:treeNode expandedIcon="ui-icon ui-icon-folder-open"
            collapsedIcon="ui-icon ui-icon-folder-collapsed">
            <h:outputText value="#{doc.name}"/>
      </p:treeNode>

      <p:treeNode type="document" icon="ui-icon ui-icon-document">
            <h:outputText value="#{doc.name}" />
      </p:treeNode>

      <p:treeNode type="picture" icon="ui-icon ui-icon-image">
            <h:outputText value="#{doc.name}" />
      </p:treeNode>

      <p:treeNode type="mp3" icon="ui-icon ui-icon-video">
            <h:outputText value="#{doc.name}" />
      </p:treeNode>
</p:tree>
```

```
public class Bean {

  private TreeNode root;

  public Bean() {
    root = new TreeNode("root", null);

    TreeNode documents = new TreeNode("Documents", root);
    TreeNode pictures = new TreeNode("Pictures", root);
    TreeNode music = new TreeNode("Music", root);

    TreeNode work = new TreeNode("Work", documents);
    TreeNode primefaces = new TreeNode("PrimeFaces", documents);

    //Documents
    TreeNode expenses = new TreeNode("document", "Expenses.doc", work);
    TreeNode resume = new TreeNode("document", "Resume.doc", work);
    TreeNode refdoc = new TreeNode("document", "RefDoc.pages", primefaces);

    //Pictures
    TreeNode barca = new TreeNode("picture", "barcelona.jpg", pictures);
    TreeNode primelogo = new TreeNode("picture", "logo.jpg", pictures);
    TreeNode optimus = new TreeNode("picture", "optimus.png", pictures);

    //Music
    TreeNode turkish = new TreeNode("Turkish", music);

    TreeNode cemKaraca = new TreeNode("Cem Karaca", turkish);
    TreeNode erkinKoray = new TreeNode("Erkin Koray", turkish);
    TreeNode mogollar = new TreeNode("Mogollar", turkish);

    TreeNode nemalacak = new TreeNode("mp3", "Nem Alacak Felek Benim", cemKaraca);
    TreeNode resimdeki = new TreeNode("mp3", "Resimdeki Goz Yaslari", cemKaraca);

    TreeNode copculer = new TreeNode("mp3", "Copculer", erkinKoray);
    TreeNode oylebirgecer = new TreeNode("mp3", "Oyle Bir Gecer", erkinKoray);

    TreeNode toprakana = new TreeNode("mp3", "Toprak Ana", mogollar);
    TreeNode bisiyapmali = new TreeNode("mp3", "Bisi Yapmali", mogollar);
  }

  public TreeNode getRoot() {
      return root;
  }
}
```

Integration between a TreeNode and a p:treeNode is the type attribute, for example music files in document explorer are represented using TreeNodes with type "mp3", there's also a p:treeNode component with same type "mp3". This results in rendering all music nodes using that particular p:treeNode representation which displays a note icon. Similarly document and pictures have their own p:treeNode representations.

Folders on the other hand have two states whose icons are defined by *expandedIcon* and *collapsedIcon* attributes.

Ajax Behavior Events

Tree provides various ajax behavior events.

Event	Listener Parameter	Fired
expand	org.primefaces.event.NodeExpandEvent	When a node is expanded.
collapse	org.primefaces.event.NodeCollapseEvent	When a node is collapsed.
select	org.primefaces.event.NodeSelectEvent	When a node is selected.
collapse	org.primefaces.event.NodeUnselectEvent	When a node is unselected.

Following tree has three listeners;

```
<p:tree value="#{treeBean.model}" dynamic="true">
      <p:ajax event="select" listener="#{treeBean.onNodeSelect}" />
      <p:ajax event="expand" listener="#{treeBean.onNodeExpand}" />
      <p:ajax event="collapse" listener="#{treeBean.onNodeCollapse}" />
      ...
</p:tree>
```

```
public void onNodeSelect(NodeSelectEvent event) {
      String node = event.getTreeNode().getData().toString();
}

public void onNodeExpand(NodeExpandEvent event) {
      String node = event.getTreeNode().getData().toString();
}

public void onNodeCollapse(NodeCollapseEvent event) {
      String node = event.getTreeNode().getData().toString();
}
```

Event listeners are also useful when dealing with huge amount of data. The idea for implementing such a use case would be providing only the root and child nodes to the tree, use event listeners to get the selected node and add new nodes to that particular tree at runtime.

Selection

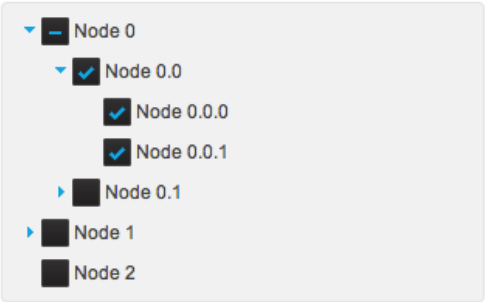
Node selection is a built-in feature of tree and it supports three different modes. Selection should be a TreeNode for single case and an array of TreeNodes for multiple and checkbox cases, tree finds the selected nodes and assign them to your selection model.

single: Only one at a time can be selected, selection should be a TreeNode reference.
multiple: Multiple nodes can be selected, selection should be a TreeNode[] reference.
checkbox: Multiple selection is done with checkbox UI, selection should be a TreeNode[] reference.

```
<p:tree value="#{treeBean.root}" var="node"
            selectionMode="checkbox"
            selection="#{treeBean.selectedNodes}">
      <p:treeNode>
            <h:outputText value="#{node}"/>
      </p:treeNode>
</p:tree>
```

```
public class TreeBean {

      private TreeNode root;

      private TreeNode[] selectedNodes;

      public TreeBean() {
            root = new TreeNode("Root", null);
            //populate nodes
      }

      //getters and setters
}
```

That's it, now the checkbox based tree looks like below. When the form is submitted with a command component like a button, selected nodes will be populated in selectedNodes property of TreeBean.



Node Caching

When caching is turned on by default, dynamically loaded nodes will be kept in memory so re-expanding a node will not trigger a server side request. In case it's set to false, collapsing the node will remove the children and expanding it later causes the children nodes to be fetched from server again.

Handling Node Click

If you need to execute custom javascript when a treenode is clicked, use the *onNodeClick* attribute. Your javascript method will be invoked with passing the html element of the node. In case you have datakey defined, you can access datakey on client side by using node.attr('data-datakey') that represents the data represented by the backing tree model.

ContextMenu

Tree has special integration with context menu, you can even match different context menus with different tree nodes using *nodeType* option of context menu that matches the tree node type.

Skinning

Tree resides in a container element which *style* and *styleClass* options apply.

Following is the list of structural style classes;

Style Class	Applies
.ui-tree	Main container
.ui-tree-nodes	Child nodes container
.ui-tree-node	Tree node
.ui-tree-node-content	Tree node content
.ui-tree-icon	Tree node icon
.ui-tree-node-label	Tree node label

As skinning style classes are global, see the main Skinning section for more information.

3.121 TreeNode

TreeNode is used with Tree component to represent a node in tree.

Info

Tag	**treeNode**
Component Class	**org.primefaces.component.tree.UITreeNode**
Component Type	**org.primefaces.component.UITreeNode**
Component Family	**org.primefaces.component**

Attributes

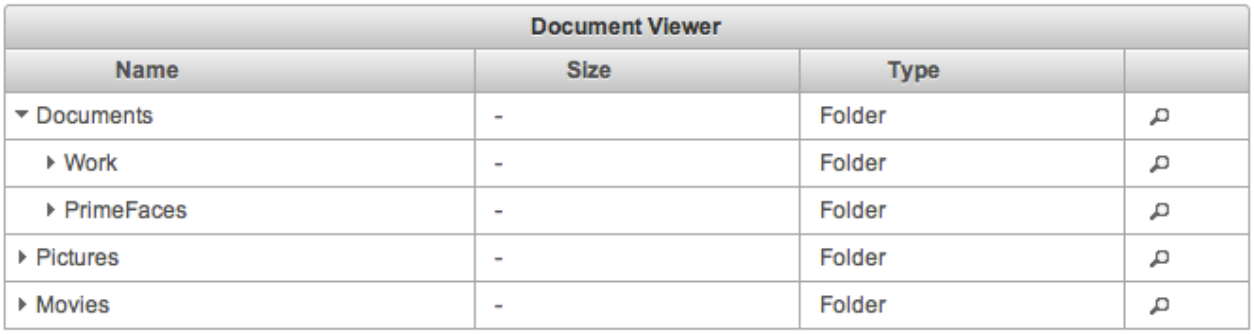
Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
type	default	String	Type of the tree node
styleClass	null	String	Style class to apply a particular tree node type
icon	null	String	Icon of the node.
expandedIcon	null	String	Expanded icon of the node.
collapsedIcon	null	String	Collapsed icon of the node.

Getting started with the TreeNode

TreeNode is used by Tree and TreeTable components, refer to sections of these components for more information.

3.122 TreeTable

Treetable is is used for displaying hierarchical data in tabular format.



Info

Tag	**treeTable**
Component Class	**org.primefaces.component.treetable.TreeTable**
Component Type	**org.primefaces.component.TreeTable**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.TreeTableRenderer**
Renderer Class	**org.primefaces.component.treetable.TreeTableRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	null	Object	A TreeNode instance as the backing model.
var	null	String	Name of the request-scoped variable used to refer each treenode.
widgetVar	null	String	Name of the client side widget.
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
selection	null	Object	Selection reference.

Name	Default	Type	Description
selectionMode	null	String	Type of selection mode.
scrollable	FALSE	Boolean	Whether or not the data should be scrollable.
scrollHeight	null	Integer	Height of scrollable data.
scrollWidth	null	Integer	Width of scrollable data.
tableStyle	null	String	Inline style of the table element.
tableStyleClass	null	String	Style class of the table element.

Getting started with the TreeTable

Similar to the Tree, TreeTable is populated with an *org.primefaces.model.TreeNode* instance that corresponds to the root node. TreeNode API has a hierarchical data structure and represents the data to be populated in tree. For an example, model to be displayed is a collection of documents.

```
public class Document {

      private String name;
      private String size;
      private String type;

      //getters, setters
}
```

```
<p:treeTable value="#{bean.root}" var="document">
      <p:column>
            <f:facet name="header">
                  Name
            </f:facet>
            <h:outputText value="#{document.name}" />
      </p:column>

      <p:column>
            <f:facet name="header">
                  Size
            </f:facet>
            <h:outputText value="#{document.size}" />
      </p:column>

      <p:column>
            <f:facet name="header">
                  Type
            </f:facet>
            <h:outputText value="#{document.type}" />
      </p:column>
</p:treeTable>
```

Backing model is same as the documents example in tree.

Selection

Node selection is a built-in feature of tree and it supports two different modes. Selection should be a TreeNode for single case and an array of TreeNodes for multiple case, tree finds the selected nodes and assign them to your selection model.

single: Only one at a time can be selected, selection should be a TreeNode reference.
multiple: Multiple nodes can be selected, selection should be a TreeNode[] reference.

Ajax Behavior Events

TreeTable provides various ajax behavior events to respond user actions.

Event	Listener Parameter	Fired
expand	org.primefaces.event.NodeExpandEvent	When a node is expanded.
collapse	org.primefaces.event.NodeCollapseEvent	When a node is collapsed.
select	org.primefaces.event.NodeSelectEvent	When a node is selected.
collapse	org.primefaces.event.NodeUnselectEvent	When a node is unselected.
colResize	org.primefaces.event.ColumnResizeEvent	When a column is resized.

ContextMenu

TreeTable has special integration with context menu, you can even match different context menus with different tree nodes using *nodeType* option of context menu that matches the tree node type.

Skinning

TreeTable content resides in a container element which style and styleClass attributes apply. Following is the list of structural style classes;

Class	Applies
.ui-treetable	Main container element.
.ui-treetable-header	Header of treetable.
.ui-treetable-data	Body element of the table containing data
.ui-tt-c	Each cell in treetable.

As skinning style classes are global, see the main Skinning section for more information.

3.123 Watermark

Watermark displays a hint on an input field.

Info

Tag	**watermark**
Component Class	**org.primefaces.component.watermark.Watermark**
Component Type	**org.primefaces.component.Watermark**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.WatermarkRenderer**
Renderer Class	**org.primefaces.component.watermark.WatermarkRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
value	0	Integer	Text of watermark.
for	null	String	Id of the component to attach the watermark
forElement	null	String	jQuery selector to attach the watermark

Getting started with Watermark

Watermark requires a target of the input component, one way is to use for attribute.

```
<h:inputText id="txt" value="#{bean.searchKeyword}" />
<p:watermark for="txt" value="Search with a keyword" />
```

Form Submissions

Watermark is set as the text of an input field which shouldn't be sent to the server when an enclosing for is submitted. This would result in updating bean properties with watermark values. Watermark component is clever enough to handle this case, by default in non-ajax form submissions, watermarks are cleared. However ajax submissions requires a little manual effort.

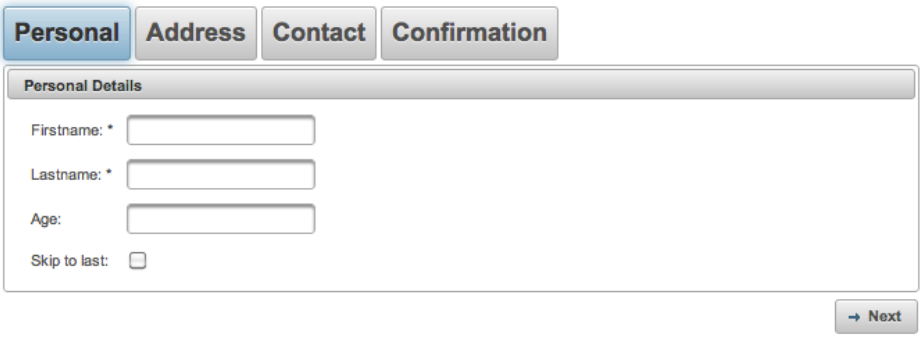
```
<h:inputText id="txt" value="#{bean.searchKeyword}" />

<p:watermark for="txt" value="Search with a keyword" />

<h:commandButton value="Submit" />
<p:commandButton value="Submit" onclick="PrimeFaces.cleanWatermarks()"
                                oncomplete="PrimeFaces.showWatermarks()" />
```

Skinning

There's only one css style class applying watermark which is '*.ui-watermark*', you can override this class to bring in your own style. Note that this style class is not applied when watermark uses html5 placeholder if available.

3.124 Wizard

Wizard provides an ajax enhanced UI to implement a workflow easily in a single page. Wizard consists of several child tab components where each tab represents a step in the process.



Info

Tag	**wizard**
Component Class	**org.primefaces.component.wizard.Wizard**
Component Type	**org.primefaces.component.Wizard**
Component Family	**org.primefaces.component**
Renderer Type	**org.primefaces.component.WizardRenderer**
Renderer Class	**org.primefaces.component.wizard.WizardRenderer**

Attributes

Name	Default	Type	Description
id	null	String	Unique identifier of the component.
rendered	TRUE	Boolean	Boolean value to specify the rendering of the component, when set to false component will not be rendered.
binding	null	Object	An el expression that maps to a server side UIComponent instance in a backing bean
step	0	String	Id of the current step in flow
style	null	String	Style of the main wizard container element.
styleClass	null	String	Style class of the main wizard container element.
flowListener	null	MethodExpr	Server side listener to invoke when wizard attempts to go forward or back.
showNavBar	TRUE	Boolean	Specifies visibility of default navigator arrows.
showStepStatus	TRUE	Boolean	Specifies visibility of default step title bar.

Name	Default	Type	Description
onback	null	String	Javascript event handler to be invoked when flow goes back.
onnext	null	String	Javascript event handler to be invoked when flow goes forward.
nextLabel	null	String	Label of next navigation button.
backLabel	null	String	Label of back navigation button.
widgetVar	null	String	Name of the client side widget

Getting Started with Wizard

Each step in the flow is represented with a tab. As an example following wizard is used to create a new user in a total of 4 steps where last step is for confirmation of the information provided in first 3 steps. To begin with create your backing bean, it's important that the bean lives across multiple requests so avoid a request scope bean. Optimal scope for wizard is viewScope.

```
public class UserWizard {

      private User user = new User();

      public User getUser() {
            return user;
      }

      public void setUser(User user) {
            this.user = user;
      }

      public void save(ActionEvent actionEvent) {
            //Persist user
            FacesMessage msg = new FacesMessage("Successful",
                        "Welcome :" + user.getFirstname());
            FacesContext.getCurrentInstance().addMessage(null, msg);
      }
}
```

User is a simple pojo with properties such as firstname, lastname, email and etc. Following wizard requires 3 steps to get the user data; Personal Details, Address Details and Contact Details. Note that last tab contains read-only data for confirmation and the submit button.

```
<h:form>

 <p:wizard>
   <p:tab id="personal">
        <p:panel header="Personal Details">

          <h:messages errorClass="error"/>

          <h:panelGrid columns="2">
             <h:outputText value="Firstname: *" />
             <h:inputText value="#{userWizard.user.firstname}" required="true"/>

             <h:outputText value="Lastname: *" />
             <h:inputText value="#{userWizard.user.lastname}" required="true"/>

             <h:outputText value="Age: " />
             <h:inputText value="#{userWizard.user.age}" />
          </h:panelGrid>
        </p:panel>
   </p:tab>

   <p:tab id="address">
     <p:panel header="Adress Details">

        <h:messages errorClass="error"/>

        <h:panelGrid columns="2" columnClasses="label, value">
            <h:outputText value="Street: " />
            <h:inputText value="#{userWizard.user.street}" />

            <h:outputText value="Postal Code: " />
            <h:inputText value="#{userWizard.user.postalCode}" />

            <h:outputText value="City: " />
            <h:inputText value="#{userWizard.user.city}" />
        </h:panelGrid>
      </p:panel>
   </p:tab>

   <p:tab id="contact">
     <p:panel header="Contact Information">

        <h:messages errorClass="error"/>

        <h:panelGrid columns="2">
            <h:outputText value="Email: *" />
            <h:inputText value="#{userWizard.user.email}" required="true"/>

            <h:outputText value="Phone: " />
            <h:inputText value="#{userWizard.user.phone}"/>

            <h:outputText value="Additional Info: " />
            <h:inputText value="#{userWizard.user.info}"/>
        </h:panelGrid>
     </p:panel>
   </p:tab>
```

```
      <p:tab id="confirm">
        <p:panel header="Confirmation">

          <h:panelGrid id="confirmation" columns="6">
            <h:outputText value="Firstname: " />
            <h:outputText value="#{userWizard.user.firstname}"/>

            <h:outputText value="Lastname: " />
            <h:outputText value="#{userWizard.user.lastname}"/>

            <h:outputText value="Age: " />
            <h:outputText value="#{userWizard.user.age}" />

            <h:outputText value="Street: " />
            <h:outputText value="#{userWizard.user.street}" />

            <h:outputText value="Postal Code: " />
            <h:outputText value="#{userWizard.user.postalCode}"/>

            <h:outputText value="City: " />
            <h:outputText value="#{userWizard.user.city}"/>

            <h:outputText value="Email: " />
            <h:outputText value="#{userWizard.user.email}" />

            <h:outputText value="Phone " />
            <h:outputText value="#{userWizard.user.phone}"/>

            <h:outputText value="Info: " />
            <h:outputText value="#{userWizard.user.info}"/>

            <h:outputText />
            <h:outputText />
      </h:panelGrid>

      <p:commandButton value="Submit" actionListener="#{userWizard.save}" />

      </p:panel>
   </p:tab>

 </p:wizard>
</h:form>
```

AJAX and Partial Validations

Switching between steps is based on ajax, meaning each step is loaded dynamically with ajax. Partial validation is also built-in, by this way when you click next, only the current step is validated, if the current step is valid, next tab's contents are loaded with ajax. Validations are not executed when flow goes back.

Navigations

Wizard provides two icons to interact with; next and prev. Please see the skinning wizard section to know more about how to change the look and feel of a wizard.

Custom UI

By default wizard displays right and left arrows to navigate between steps, if you need to come up with your own UI, set *showNavBar* to false and use the provided the client side api.

```
<p:wizard showNavBar="false" widgetVar="wiz">
      ...
</p:wizard>

<h:outputLink value="#" onclick="wiz.next();">Next</h:outputLink>
<h:outputLink value="#" onclick="wiz.back();">Back</h:outputLink>
```

Ajax FlowListener

If you'd like get notified on server side when wizard attempts to go back or forward, define a flowListener.

```
<p:wizard flowListener="#{userWizard.handleFlow}">
      ...
</p:wizard>
```

```
public String handleFlow(FlowEvent event) {
      String currentStepId = event.getCurrentStep();
      String stepToGo = event.getNextStep();

      if(skip)
            return "confirm";
      else
            return event.getNextStep();
}
```

Steps here are simply the ids of tab, by using a flowListener you can decide which step to display next so wizard does not need to be linear always. If you need to update other component(s) on page within a flow, use *RequestContext.update(String clientId)* api.

Client Side Callbacks

Wizard is equipped with onback and onnext attributes, in case you need to execute custom javascript after wizard goes back or forth. You just need to provide the names of javascript functions as the values of these attributes.

```
<p:wizard onnext="alert('Next')" onback="alert('Back')">
      ...
</p:wizard>
```

Client Side API

Widget: *PrimeFaces.widget.Wizard*

Method	Params	Return Type	Description
next()	-	void	Proceeds to next step.
back()	-	void	Goes back in flow.
getStepIndex()	-	Number	Returns the index of current step.
showNextNav()	-	void	Shows next button.
hideNextNav()	-	void	Hides next button.
showBackNav()	-	void	Shows back button.
hideBackNav()	-	void	Hides back button.

Skinning

Wizard resides in a container element that *style* and *styleClass* attributes apply.

Following is the list of structural css classes.

Selector	Applies
.ui-wizard	Main container element.
.ui-wizard-content	Container element of content.
.ui-wizard-step-titles	Container of step titles.
.ui-wizard-step-title	Each step title.
.ui-wizard-navbar	Container of navigation controls.
.ui-wizard-nav-back	Back navigation control.
.ui-wizard-nav-next	Forward navigation control.

As skinning style classes are global, see the main Skinning section for more information.

4. Partial Rendering and Processing

PrimeFaces provides a partial rendering and view processing feature based on standard JSF 2 APIs to enable choosing what to process in JSF lifecyle and what to render in the end with ajax.

4.1 Partial Rendering

In addition to components like autoComplete, datatable, slider with built-in ajax capabilities, PrimeFaces also provides a generic PPR (Partial Page Rendering) mechanism to update JSF components with ajax. Several components are equipped with the common PPR attributes (e.g. update, process, onstart, oncomplete).

4.1.1 Infrastructure

PrimeFaces Ajax Framework is based on standard server side APIs of JSF 2. There are no additional artfacts like custom AjaxViewRoot, AjaxStateManager, AjaxViewHandler, Servlet Filters, HtmlParsers, PhaseListeners and so on. PrimeFaces aims to keep it clean, fast and lightweight.

On client side rather than using client side API implementations of JSF implementations like Mojarra and MyFaces, PrimeFaces scripts are based on the most popular javascript library; jQuery which is far more tested, stable regarding ajax, dom handling, dom tree traversing than a JSF implementations scripts.

4.1.2 Using IDs

Getting Started

When using PPR you need to specify which component(s) to update with ajax. If the component that triggers PPR request is at the same namingcontainer (eg. form) with the component(s) it renders, you can use the server ids directly. In this section although we'll be using commandButton, same applies to every component that's capable of PPR such as commandLink, poll, remoteCommand and etc.

```
<h:form>
      <p:commandButton update="display" />

      <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

PrependId

Setting prependId setting of a form has no effect on how PPR is used.

```
<h:form prependId="false">
      <p:commandButton update="display" />

      <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

ClientId

It is also possible to define the client id of the component to update.

```
<h:form id="myform">
      <p:commandButton update="myform:display" />
      <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

Different NamingContainers

If your page has different naming containers (e.g. two forms), you also need to add the container id to search expression so that PPR can handle requests that are triggered inside a namingcontainer that updates another namingcontainer. Following is the suggested way using separator char as a prefix, note that this uses same search algorithm as standard JSF 2 implementation;

```
<h:form id="form1">

      <p:commandButton update=":form2:display" />

</h:form>

<h:form id="form2">

      <h:outputText id="display" value="#{bean.value}"/>

</h:form>
```

Please read **findComponent** algorithm described in link below used by both JSF core and PrimeFaces to fully understand how component referencing works.

```
http://docs.oracle.com/javaee/6/api/javax/faces/component/UIComponent.html
```

JSF h:form, datatable, composite components are naming containers, in addition tabView, accordionPanel, dataTable, dataGrid, dataList, carousel, galleria, ring, sheet and subTable are PrimeFaces component that implement NamingContainer.

Multiple Components

Multiple Components to update can be specified with providing a list of ids separated by a comma, whitespace or even both.

```
<h:form>

      <p:commandButton update="display1,display2" />
      <p:commandButton update="display1 display2" />

      <h:outputText id="display1" value="#{bean.value1}"/>

      <h:outputText id="display2" value="#{bean.value2}"/>

</h:form>
```

Keywords

There are a couple of reserved keywords which serve as helpers.

Keyword	Description
@this	Component that triggers the PPR is updated
@parent	Parent of the PPR trigger is updated.
@form	Encapsulating form of the PPR trigger is updated
@none	PPR does not change the DOM with ajax response.
@all	Whole document is updated as in non-ajax requests.

Example below updates the whole form.

```
<h:form>

      <p:commandButton update="@form" />

      <h:outputText value="#{bean.value}"/>

</h:form>
```

Keywords can also be used together with explicit ids, so update="@form, display" is also supported.

4.1.3 Notifying Users

ajaxStatus is the component to notify the users about the status of **global** ajax requests. See the ajaxStatus section to get more information about the component.

Global vs Non-Global

By default ajax requests are global, meaning if there is an ajaxStatus component present on page, it is triggered.

If you want to do a "silent" request not to trigger ajaxStatus instead, set global to false. An example with commandButton would be;

```
<p:commandButton value="Silent" global="false" />

<p:commandButton value="Notify" global="true" />
```

4.1.4 Bits&Pieces

PrimeFaces Ajax Javascript API

See the javascript section to learn more about the PrimeFaces Javascript Ajax API.

4.2 Partial Processing

In Partial Page Rendering, only specified components are rendered, similarly in Partial Processing only defined components are processed. Processing means executing Apply Request Values, Process Validations, Update Model and Invoke Application JSF lifecycle phases only on defined components.

This feature is a simple but powerful enough to do group validations, avoiding validating unwanted components, eliminating need of using immediate and many more use cases. Various components such as commandButton, commandLink are equipped with process attribute, in examples we'll be using commandButton.

4.2.1 Partial Validation

A common use case of partial process is doing partial validations, suppose you have a simple contact form with two dropdown components for selecting city and suburb, also there's an inputText which is required. When city is selected, related suburbs of the selected city is populated in suburb dropdown.

```
<h:form>

  <h:selectOneMenu id="cities" value="#{bean.city}">
        <f:selectItems value="#{bean.cityChoices}" />
        <p:ajax listener="#{bean.populateSuburbs}" update="suburbs"
            process="@all"/>
  </h:selectOneMenu>

  <h:selectOneMenu id="suburbs" value="#{bean.suburb}">
      <f:selectItems value="#{bean.suburbChoices}" />
  </h:selectOneMenu>

  <h:inputText value="#{bean.email}" required="true"/>

</h:form>
```

When the city dropdown is changed an ajax request is sent to execute populateSuburbs method which populates suburbChoices and finally update the suburbs dropdown. Problem is populateSuburbs method will not be executed as lifecycle will stop after process validations phase to jump render response as email input is not provided. Reason is p:ajax has @all as the value stating to process every component on page but there is no need to process the inputText.

The solution is to define what to process in p:ajax. As we're just making a city change request, only processing that should happen is cities dropdown.

```
<h:form>
  <h:selectOneMenu id="cities" value="#{bean.city}">
        <f:selectItems value="#{bean.cityChoices}" />
        <p:ajax actionListener="#{bean.populateSuburbs}"
             event="change" update="suburbs" process="@this"/>
  </h:selectOneMenu>

  <h:selectOneMenu id="suburbs" value="#{bean.suburb}">
      <f:selectItems value="#{bean.suburbChoices}" />
  </h:selectOneMenu>

  <h:inputText value="#{bean.email}" required="true"/>
</h:form>
```

That is it, now populateSuburbs method will be called and suburbs list will be populated. Note that default value for process option is @this already for p:ajax as stated in AjaxBehavior documentation, it is explicitly defined here to give a better understanding of how partial processing works.

4.2.2 Keywords

Just like updates, partial processing also supports keywords.

Keyword	Description
@this	Component that triggers the PPR is processed.
@parent	Parent of the PPR trigger is processed.
@form	Encapsulating form of the PPR trigger is processed
@none	No component is processed, useful to revert changes to form.
@all	Whole component tree is processed just like a regular request.

Important point to note is, when a component is specified to process partially, children of this component is processed as well. So for example if you specify a panel, all children of that panel would be processed in addition to the panel itself.

```
 <p:commandButton process="panel" />

<p:panel id="panel">
      //Children
</p:panel>
```

4.2.3 Using Ids

Partial Process uses the same technique applied in PPR to specify component identifiers to process. See section 5.1.2 for more information about how to define ids in process specification using commas and whitespaces.

4.3 PFS (PrimeFaces Selectors)

PFS integrates jQuery Selector API with JSF component referencing model, so for partial update and process, referencing JSF components can be done using jQuery Selector API instead of regular JSF model which is based on UIComponent.findComponent. Here are some examples;

Update all forms

```
update="@(form)"
```

Update first form

```
update="@(form:first)"
```

Update all components that has styleClass named mystyle

```
update="@(.mystyle)"
```

Update and process all inputs

```
update="@(:input)" process="@(:input)"
```

Update all datatables

```
update="@(.ui-datatable)"
```

Process input components inside any panel and update all panels

```
process="@(.ui-panel :input)" update="@(.ui-panel)"
```

Process input components but not select components

```
process="@(:input:not(select))"
```

Update input components that are disabled

```
update="@(:input:disabled)"
```

PFS can be used with regular component referencing as well;

```
update="compId :form:compId @(:input)"
```

PFS provides an alternative, flexible, grouping based approach to reference components to partially process and update. There is less CPU server load compared to regular referencing because JSF component tree is not traversed on server side to find a component and figure out the client id as PFS is implemented on client side by looking at dom tree. Another advantage is avoiding naming container limitations, just remember the times you've faced with cannot find component exception since the component you are looking for is in a different naming container like a form or a datatable. PFS can help you out in tricky situations by following jQuery's "write less do more" style.

For full reference of jQuery selector api, see;

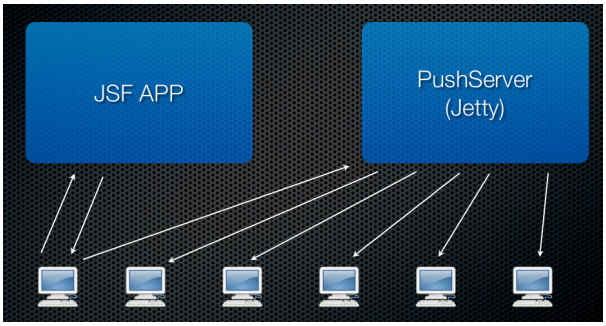
http://api.jquery.com/category/selectors/

5. PrimeFaces Mobile

PrimeFaces Mobile is a separate project with it's own release cycle and documentation. Please consult Mobile User's Guide for more information.

6. PrimeFaces Push

PrimePush enables implementing push based use-cases powered by WebSockets. PushServer and JSF application are two different applications. Pushed data from JSF app is send to the push server which is then pushed to all connected clients.



6.1 Setup

Push Server

PrimeFaces Push uses a servlet as a dispatcher. This servlet should be in a different application than the JSF application and at the moment can only be deployed on jetty server.

```
<servlet>
      <servlet-name>Push Servlet</servlet-name>
      <servlet-class>org.primefaces.push.PushServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
      <init-param>
            <param-name>channels</param-name>
            <param-value>chat,counter</param-value>
      </init-param>
</servlet>

<servlet-mapping>
      <servlet-name>Push Servlet</servlet-name>
      <url-pattern>/prime-push/*</url-pattern>
</servlet-mapping>
```

channels configuration defines the topic names that push server can publish to.

URL Configuration

The JSF application needs to define the push server url to send messages.

```
<context-param>
      <param-name>primefaces.PUSH_SERVER_URL</param-name>
      <param-value>ws://url_to_push_server</param-value>
</context-param>
```

6.2 Push API

JSF app can push data to the server by using the RequestContext API.

```
/**
 * @param channel Unique name of communication channel
 * @param data Information to be pushed to the subscribers as json
 */
RequestContext.push(String channel, Object data);
```

6.3 Push Component

<p:push /> is a PrimeFaces component that handles the connection between the server and the browser, it has two attributes you need to define.

```
<p:push channel="chat" onmessage="handlePublish"/>
```

channel: Name of the channel to connect and listen.
onpublish: Javascript event handler to be called when server sends data.

6.4 Samples

6.4.1 Counter

Counter is a global counter where each button click increments the count value and new value is pushed to all subscribers.

```
public class GlobalCounterBean implements Serializable{

      private int count;

      public int getCount() {
            return count;
      }

      public void setCount(int count) {
            this.count = count;
      }

      public synchronized void increment() {
            count++;
            RequestContext.getCurrentInstance().push("counter", count);
      }
}
```

```
<h:form>
      <h:outputText value="#{globalCounter.count}" styleClass="display"/>

      <p:commandButton value="Click" actionListener="#{globalCounter.increment}" />
</h:form>

<p:push onmessage="handleMessage" channel="counter" />
```

```
<script type="text/javascript">
      function handleMessage(evt, data) {
            $('.display').html(data);
      }
</script>
```

When the button is clicked, JSF button invokes the increment method and RequestContext API forwards the pushed data to the counter channel of the push server, then push server forwards this data to all subscribers. Finally handleMessage javascript function is called where data parameter is the new count integer.

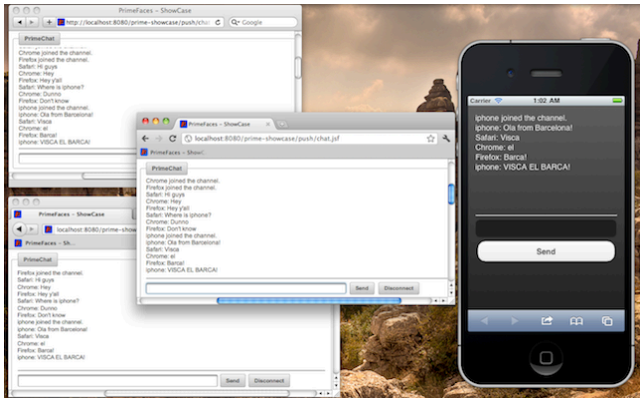
6.4.2 Chat

Chat is a simple p2p messaging implementation.

```
public class ChatBean implements Serializable {

      private String message;
      private String username;
      private boolean loggedIn;

      public void send(ActionEvent event) {
            RequestContext.getCurrentInstance().push("chat",
                                    username + ": " + message);
            message = null;
      }

      public void login(ActionEvent event) {
            loggedIn = true;
            RequestContext.getCurrentInstance().push("chat",
                                          username + " has logged in.");
      }

      //getters&setters
}
```

```
<h:form>
 <p:fieldset id="container" legend="PrimeChat">
   <h:panelGroup rendered="#{chatController.loggedIn}" >
     <p:outputPanel layout="block" style="width:600px;height:200px;overflow:auto"
                                    styleClass="chatContent" />

      <p:separator />

      <p:inputText value="#{chatController.message}" styleClass="messageInput" />
      <p:spacer width="5" />
      <p:commandButton value="Send" actionListener="#{chatController.send}"
            global="false" oncomplete="$('.messageInput').val('').focus()"/>
      <p:spacer width="5" />
      <p:commandButton value="Disconnect"
            actionListener="#{chatController.disconnect}" global="false"
            oncomplete="chatAgent.close()" update="container" />
   </h:panelGroup>

   <h:panelGroup rendered="#{not chatController.loggedIn}" >
     Username: <p:inputText value="#{chatController.username}" />

      <p:spacer width="5" />
      <p:commandButton value="Login"
            actionListener="#{chatController.login}" update="container"
            image="ui-icon ui-icon-person"/>
   </h:panelGroup>
 </p:fieldset>
</h:form>

<p:push onmessage="handleMessage" channel="chat" widgetVar="chatAgent" />
```

```
<script type="text/javascript">
      function handleMessage(evt, data) {
            var chatContent = $('.chatContent'); c
            hatContent.append(data + '');

            //keep scroll chatContent.scrollTop(chatContent.height());
      }
</script>
```



PrimeFaces Push currently provides P2P communication and next feature to support is JMS integration.

7. Javascript API

PrimeFaces renders unobstrusive javascript which cleanly seperates behavior from the html. Client side engine is powered by jQuery version 1.7.2 which is the latest at the time of the writing.

7.1 PrimeFaces Namespace

PrimeFaces is the main javascript object providing utilities and namespace.

Method	Description
escapeClientId(id)	Escaped JSF ids with semi colon to work with jQuery.
addSubmitParam(el, name, param)	Adds request parameters dynamically to the element.
getCookie(name)	Returns cookie with given name.
setCookie(name, value)	Sets a cookie with given nam and value.
skinInput(input)	Progressively enhances an input element with theming.
info(msg), debug(msg), warn(msg), error(msg)	Client side log API.
changeTheme(theme)	Changes theme on the fly with no page refresh.
cleanWatermarks()	Watermark component extension, cleans all watermarks on page before submitting the form.
showWatermarks()	Shows watermarks on form.

To be compatible with other javascript entities on a page, PrimeFaces defines two javascript namespaces;

PrimeFaces.widget.*

Contains custom PrimeFaces widgets like;

- PrimeFaces.widget.DataTable
- PrimeFaces.widget.Tree
- PrimeFaces.widget.Poll
- and more...

Most of the components have a corresponding client side widget with same name.

PrimeFaces.ajax.*

PrimeFaces.ajax namespace contains the ajax API which is described in next section.

7.2 Ajax API

PrimeFaces Ajax Javascript API is powered by jQuery and optimized for JSF. Whole API consists of three properly namespaced simple javascript functions.

PrimeFaces.ajax.AjaxRequest

Sends ajax requests that execute JSF lifecycle and retrieve partial output. Function signature is as follows;

```
PrimeFaces.ajax.AjaxRequest(cfg);
```

Configuration Options

Option	Description
formId	Id of the form element to serialize, if not defined parent form of source is used.
async	Flag to define whether request should go in ajax queue or not, default is false.
global	Flag to define if p:ajaxStatus should be triggered or not, default is true.
update	Component(s) to update with ajax.
process	Component(s) to process in partial request.
source	Client id of the source component causing the request.
params	Additional parameters to send in ajax request.
onstart()	Javascript callback to process before sending the ajax request, return false to cancel the request.
onsuccess(data, status, xhr, args)	Javascript callback to process when ajax request returns with success code. Takes four arguments, xml response, status code, xmlhttprequest and optional arguments provided by RequestContent API.
onerror(xhr, status, exception)	Javascript callback to process when ajax request fails. Takes three arguments, xmlhttprequest, status string and exception thrown if any.
oncomplete(xhr, status, args)	Javascript callback to process when ajax request completes. Takes three arguments, xmlhttprequest, status string and optional arguments provided by RequestContext API.

Examples

Suppose you have a JSF page called *createUser* with a simple form and some input components.

```
<h:form id="userForm">
      <h:inputText id="username" value="#{userBean.user.name}" />
      ... More components
</h:form>
```

You can post all the information in form with ajax using;

```
PrimeFaces.ajax.AjaxRequest({
            formId:'userForm'
            ,source:'userForm'
            ,process:'userForm'
      });
```

More complex example with additional options;

```
PrimeFaces.ajax.AjaxRequest({
                    formId:'userForm',
                    ,source:'userForm'
                     ,process:'userForm'
                    ,update:'msgs'
                    ,params:{
                          'param_name1':'value1',
                          'param_name2':'value2'
                    }
                     ,oncomplete:function(xhr, status) {alert('Done');}
                   });
```

We highly recommend using p:remoteComponent instead of low level javascript api as it generates the same with much less effort and less possibility to do an error.

PrimeFaces.ajax.AjaxResponse

PrimeFaces.ajax.AjaxResponse updates the specified components if any and synchronizes the client side JSF state. DOM updates are implemented using jQuery which uses a very fast algorithm.

8. Themes

PrimeFaces is integrated with powerful ThemeRoller CSS Framework. Currently there are 30+ pre-designed themes that you can preview and download from PrimeFaces theme gallery.

http://www.primefaces.org/themes.html

8.1 Applying a Theme

Applying a theme to your PrimeFaces project is very easy. Each theme is packaged as a jar file, download the theme you want to use, add it to the classpath of your application and then define primefaces.THEME context parameter at your deployment descriptor (web.xml) with the theme name as the value.

Download

Each theme is available for manual download at PrimeFaces Theme Gallery. If you are a maven user, define theme artifact as;

```
<dependency>
      <groupId>org.primefaces.themes</groupId>
      <artifactId>cupertino</artifactId>
      <version>1.0.2</version>
</dependency>
```

artifactId is the name of the theme as defined at Theme Gallery page.

Configure

Once you've downloaded the theme, configure PrimeFaces to use it.

```
<context-param>
      <param-name>primefaces.THEME</param-name>
      <param-value>aristo</param-value>
</context-param>
```

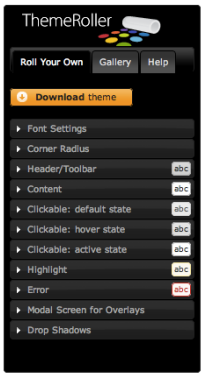
That's it, you don't need to manually add any css to your pages or anything else, PrimeFaces will handle everything for you.

In case you'd like to make the theme dynamic, define an EL expression as the param value.

```
<context-param>
      <param-name>primefaces.THEME</param-name>
      <param-value>#{loggedInUser.preferences.theme}</param-value>
</context-param>
```

8.2 Creating a New Theme

If you'd like to create your own theme instead of using the pre-defined ones, that is easy as well because ThemeRoller provides a powerful and easy to use online visual tool.



Applying your own custom theme is same as applying a pre-built theme however you need to migrate the downloaded theme files from ThemeRoller to PrimeFaces Theme Infrastructure. PrimeFaces Theme convention is the integrated way of applying your custom themes to your project, this approach requires you to create a jar file and add it to the classpath of your application. Jar file **must** have the following folder structure. You can have one or more themes in same jar.

```
- jar
        - META-INF
                - resources
                        - primefaces-yourtheme
                                - theme.css
                                - images
```

1) The theme package you've downloaded from ThemeRoller will have a css file and images folder. Make sure you have "deselect all components" option on download page so that your theme only includes skinning styles. Extract the contents of the package and rename *jquery-ui-{version}.custom.css* to *theme.css*.

2) Image references in your theme.css must also be converted to an expression that JSF resource loading can understand, example would be;

url("images/ui-bg_highlight-hard_100_f9f9f9_1x100.png")

should be;

url("#{resource['primefaces-yourtheme:images/ui-bg_highlight-hard_100_f9f9f9_1x100.png']}")

Once the jar of your theme is in classpath, you can use your theme like;

```
<context-param>
      <param-name>primefaces.THEME</param-name>
      <param-value>yourtheme</param-value>
</context-param>
```

8.3 How Themes Work

Powered by ThemeRoller, PrimeFaces separates structural css from skinning css.

Structural CSS

These style classes define the skeleton of the components and include css properties such as margin, padding, display type, dimensions and positioning.

Skinning CSS

Skinning defines the look and feel properties like colors, border colors, background images.

Skinning Selectors

ThemeRoller features a couple of skinning selectors, most important of these are;

Selector	Applies
.ui-widget	All PrimeFaces components
.ui-widget-header	Header section of a component
.ui-widget-content	Content section of a component
.ui-state-default	Default class of a clickable
.ui-state-hover	Hover class of a clickable
.ui-state-active	When a clickable is selected
.ui-state-disabled	Disabled elements.
.ui-state-highlight	Highlighted elements.
.ui-icon	An element to represent an icon.

These classes are not aware of structural css like margins and paddings, mostly they only define colors. This clean separation brings great flexibility in theming because you don't need to know each and every skinning selectors of components to change their style.

For example Panel component's header section has the *.ui-panel-titlebar* structural class, to change the color of a panel header you don't need to about this class as *.ui-widget-header* also that defines the panel colors also applies to the panel header.

8.4 Theming Tips

- Default font size of themes might be bigger than expected, to change the font-size of PrimeFaces components globally, use the .ui-widget style class. An example of smaller fonts;

```
.ui-widget, .ui-widget .ui-widget {
      font-size: 90% !important;
}
```

- When creating your own theme with themeroller tool, select one of the pre-designed themes that is close to the color scheme you want and customize that to save time.

- If you are using Apache Trinidad or JBoss RichFaces, PrimeFaces Theme Gallery includes Trinidad's Casablanca and RichFaces's BlueSky theme. You can use these themes to make PrimeFaces look like Trinidad or RichFaces components during migration.

- To change the style of a particular component instead of all components of same type use namespacing, example below demonstrates how to change header of all panels.

```
.ui-panel-titlebar {
      //css
}
```

or

```
.ui-paneltitlebar.ui-widget-header {
      //css
}
```

To apply css on a particular panel;

```
<p:panel styleClass="custom">
      ...
 </p:panel>
```

```
.custom .ui-panel-titlebar {
      //css
}
```

- Set *primefaces.THEME_FORMS* context parameter to false if you'd like to disable theming on input field components such as p:inputText, p:inputTextarea.

9. Utilities

9.1 RequestContext

RequestContext is a simple utility that provides useful goodies such as adding parameters to ajax callback functions.

RequestContext can be obtained similarly to FacesContext.

```
RequestContext requestContext = RequestContext.getCurrentInstance();
```

RequestContext API

Method	Description
isAjaxRequest()	Returns a boolean value if current request is a PrimeFaces ajax request.
addCallBackParam(String name, Object value)	Adds parameters to ajax callbacks like oncomplete.
update(String clientId);	Specifies component(s) to update at runtime.
execute(String script)	Executes script after ajax request completes.
scrollTo(String clientId)	Scrolls to the component with given clientId after ajax request completes.

Callback Parameters

There may be cases where you need values from backing beans in ajax callbacks. Suppose you have a form in a p:dialog and when the user ends interaction with form, you need to hide the dialog or if there're any validation errors, dialog needs to stay open.

Callback Parameters are serialized to JSON and provided as an argument in ajax callbacks.

```
<p:commandButton actionListener="#{bean.validate}"
      oncomplete="handleComplete(xhr, status, args)" />
```

```
public void validate() {
      //isValid = calculate isValid
      RequestContext requestContext = RequestContext.getCurrentInstance();
      requestContext.addCallbackParam("isValid", true or false);
}
```

isValid parameter will be available in handleComplete callback as;

```
<script type="text/javascript">
      function handleComplete(xhr, status, args) {
            var isValid = args.isValid;
            if(isValid)
                  dialog.hide();
      }
</script>
```

You can add as many callback parameters as you want with addCallbackParam API. Each parameter is serialized as JSON and accessible through args parameter so pojos are also supported just like primitive values.

Following example sends a pojo called User that has properties like firstname and lastname to the client.

```
public void validate() {
      //isValid = calculate isValid
      RequestContext requestContext = RequestContext.getCurrentInstance();
      requestContext.addCallbackParam("isValid", true or false);
      requestContext.addCallbackParam("user", user);
}
```

```
<script type="text/javascript">
      function handleComplete(xhr, status, args) {
            var firstname = args.user.firstname;
            var lastname = args.user.lastname;
      }
</script>
```

Default validationFailed

By default *validationFailed* callback parameter is added implicitly if JSF validation fails.

Runtime Partial Update Configuration

There may be cases where you need to define which component(s) to update at runtime rather than specifying it declaratively at compile time. addPartialUpdateTarget method is added to handle this case. In example below, button actionListener decides which part of the page to update on-the-fly.

```
<p:commandButton value="Save" actionListener="#{bean.save}" />
<p:panel id="panel"> ... </p:panel>
<p:dataTable id="table"> ... </p:panel>
```

```
public void save() {
      //boolean outcome = ...
      RequestContext requestContext = RequestContext.getCurrentInstance();

      if(outcome)
            requestContext.update("panel");
      else
            requestContext.update("table");
}
```

When the save button is clicked, depending on the outcome, you can either configure the datatable or the panel to be updated with ajax response.

Execute Javascript

RequestContext provides a way to execute javascript when the ajax request completes, this approach is easier compared to passing callback params and execute conditional javascript. Example below hides the dialog when ajax request completes;

```
public void save() {
      RequestContext requestContext = RequestContext.getCurrentInstance();

      requestContext.execute(“dialog.hide()”);
}
```

9.2 EL Functions

PrimeFaces provides built-in EL extensions that are helpers to common use cases.

Common Functions

Function	Description
component('id')	Returns clientId of the component with provided server id parameter. This function is useful if you need to work with javascript.
widgetVar('id')	Provides the widgetVar of a component.

Component

```
 <h:form id="form1">
      <h:inputText id="name" />
</h:form>

//#{p:component('name')} returns 'form1:name'
```

WidgetVar

```
 <p:dialog id="dlg">
      //contents
 </p:dialog>

<p:commandButton type="button" value="Show" onclick="#{p:widgetVar('dlg')}.show()" />
```

Page Authorization

Function	Description
ifGranted(String role)	Returns a boolean value if user has given role or not.
ifAllGranted(String roles)	Returns a boolean value if has all of the given roles or not.
ifAnyGranted(String roles)	Returns a boolean value if has any of the given roles or not.
ifNotGranted(String roles)	Returns a boolean value if has all of the given roles or not.
remoteUser()	Returns the name of the logged in user.
userPrincipal()	Returns the principal instance of the logged in user.

```
 <p:commandButton rendered="#{p:ifGranted('ROLE_ADMIN')}" />

<h:inputText disabled="#{p:ifGranted('ROLE_GUEST')}" />

<p:inputMask rendered="#{p:ifAllGranted('ROLE_EDITOR, ROLE_READER')}" />

<p:commandButton rendered="#{p:ifAnyGranted('ROLE_ADMIN, ROLE_EDITOR')}" />

<p:commandButton rendered="#{p:ifNotGranted('ROLE_GUEST')}" />

<h:outputText value="Welcome: #{p:remoteUser()}" />
```

10. Portlets

PrimeFaces supports portlet environments based on JSF 2 and Portlet 2 APIs. A portlet bridge is necessary to run a JSF application as a portlet and we've tested the bridge from portletfaces project. A kickstart example is available at PrimeFaces examples repository;

http://primefaces.googlecode.com/svn/examples/trunk/prime-portlet

10.1 Dependencies

Only necessary dependency compared to a regular PrimeFaces application is the JSF bridge, 2.0.1 is the latest version of portletfaces at the time of writing. Here's maven dependencies configuration from portlet sample.

```
<dependencies>

      <dependency>
            <groupId>javax.faces</groupId>
            <artifactId>jsf-api</artifactId>
            <version>2.0</version>
      </dependency>

      <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-impl</artifactId>
            <version>2.0.4-b09</version>
      </dependency>

      <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>2.2</version>
        </dependency>

        <dependency>
            <groupId>javax.portlet</groupId>
            <artifactId>portlet-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>org.portletfaces</groupId>
            <artifactId>portletfaces-bridge</artifactId>
            <version>2.0.1</version>
        </dependency>

</dependencies>
```

10.2 Configuration

portlet.xml

Portlet configuration file should be located under WEB-INF folder. This portlet has two modes, view and edit.

```
 <?xml version="1.0"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
version="2.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">
 <portlet>
      <portlet-name>1</portlet-name>
      <display-name>PrimeFaces Portlet</display-name>
      <portlet-class>org.portletfaces.bridge.GenericFacesPortlet</portlet-class>
      <init-param>
            <name>javax.portlet.faces.defaultViewId.view</name>
            <value>/view.xhtml</value>
      </init-param>
      <init-param>
            <name>javax.portlet.faces.defaultViewId.edit</name>
            <value>/edit.xhtml</value>
      </init-param>
      <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
            <portlet-mode>edit</portlet-mode>
      </supports>
      <portlet-info>
            <title>PrimeFaces Portlet</title>
            <short-title>PrimeFaces Portlet</short-title>
            <keywords>JSF 2.0</keywords>
      </portlet-info>
 </portlet>
</portlet-app>
```

web.xml

Faces Servlet is only necessary to initialize JSF framework internals.

```
 <?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/
j2ee/web-app_2_5.xsd" version="2.5">
      <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
</web-app>
```

faces-config.xml

An empty faces-config.xml seems to be necessary otherwise bridge is giving an error.

```
 <?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-facesconfig_2_0.xsd"
    version="2.0">

</faces-config>
```

liferay-portlet.xml

Liferay portlet configuration file is an extension to standard portlet configuration file.

```
 <?xml version="1.0"?>
<liferay-portlet-app>
      <portlet>
            <portlet-name>1</portlet-name>
            <instanceable>true</instanceable>
            <ajaxable>false</ajaxable>
      </portlet>
</liferay-portlet-app>
```

liferay-display.xml

Display configuration is used to define the location of your portlet in liferay menu.

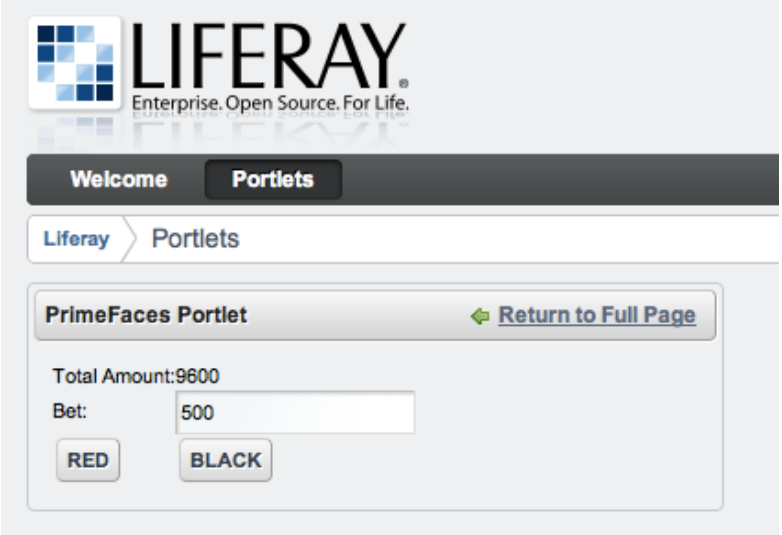
```
 <?xml version="1.0"?>
<!DOCTYPE display PUBLIC "-//Liferay//DTD Display 5.1.0//EN" "http://www.liferay.com/
dtd/liferay-display_5_1_0.dtd">

<display>
      <category name="category.sample">
            <portlet id="1" />
      </category>
</display>
```

Pages

That is it for the configuration, a sample portlet page is a partial version of the regular page to provide only the content without html and body tags.

edit.xhtml

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:p="http://primefaces.prime.com.tr/ui">

    <h:head></h:head>

    <h:form>

        <h:panelGrid id="grid" columns="2" cellpadding="10px">

            <f:facet name="header">
                <p:messages id="messages" />
            </f:facet>

            <h:outputText value="Total Amount: " />
            <h:outputText value="#{gambitController.amount}" />

            <h:outputText value="Bet:" />
            <h:inputText value="#{gambitController.bet}" />

            <p:commandButton value="RED"
                    actionListener="#{gambitController.playRed}" update="@parent" />
            <p:commandButton value="BLACK"
                    actionListener="#{gambitController.playBlack}" update="@parent" />
        </h:panelGrid>

    </h:form>

</f:view>
```



PrimeFaces Team is in contact with PortletBridge team to improve the integration.

11. Integration with Java EE

PrimeFaces is all about front-end and can be backed by your favorite enterprise application framework. Following frameworks are fully supported;

- Spring Core (JSF Centric JSF-Spring Integration)
- Spring WebFlow (Spring Centric JSF-Spring Integration)
- Spring Roo (PrimeFaces Addon)
- EJBs
- CDI

We've created sample applications to demonstrate several technology stacks involving PrimeFaces and JSF at the front layer. Source codes of these applications are available at the PrimeFaces subversion repository and they're deployed online time to time.

CDI and EJBs

PrimeFaces fully supports a JAVA EE 6 environment with CDI and EJBs.

Spring WebFlow

We as PrimeFaces team work closely with Spring WebFlow team, PrimeFaces is suggested by SpringSource as the preferred JSF component suite for SWF applications. SpringSource repository has two samples based on SWF-PrimeFaces; a small showcase and booking-faces example.

Spring ROO

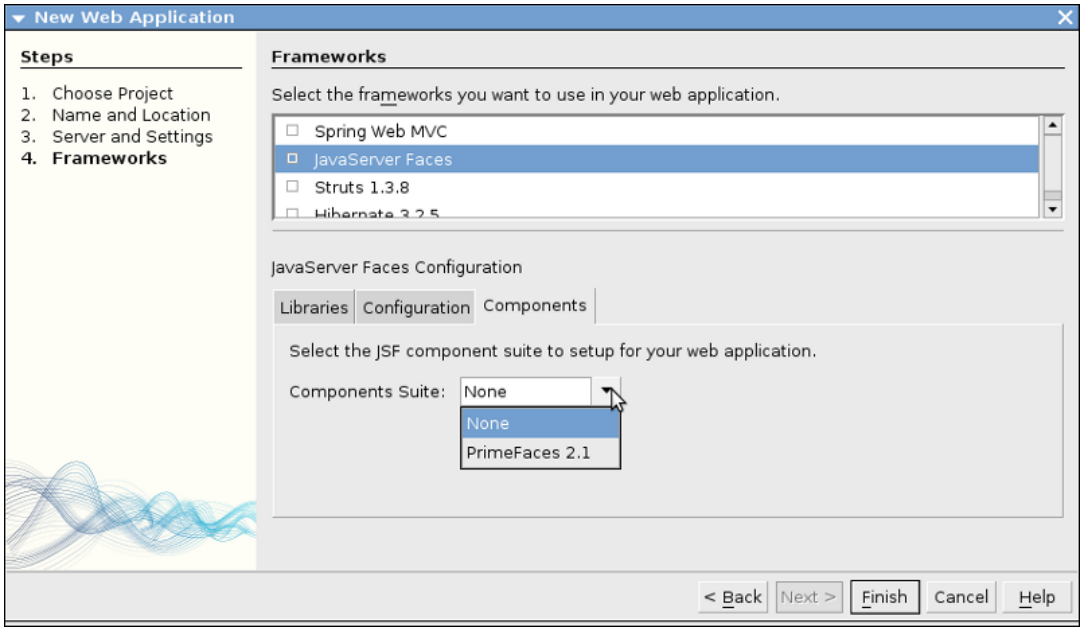
SpringSource provides an official JSF-PrimeFaces Addon.

https://jira.springsource.org/browse/ROO-516

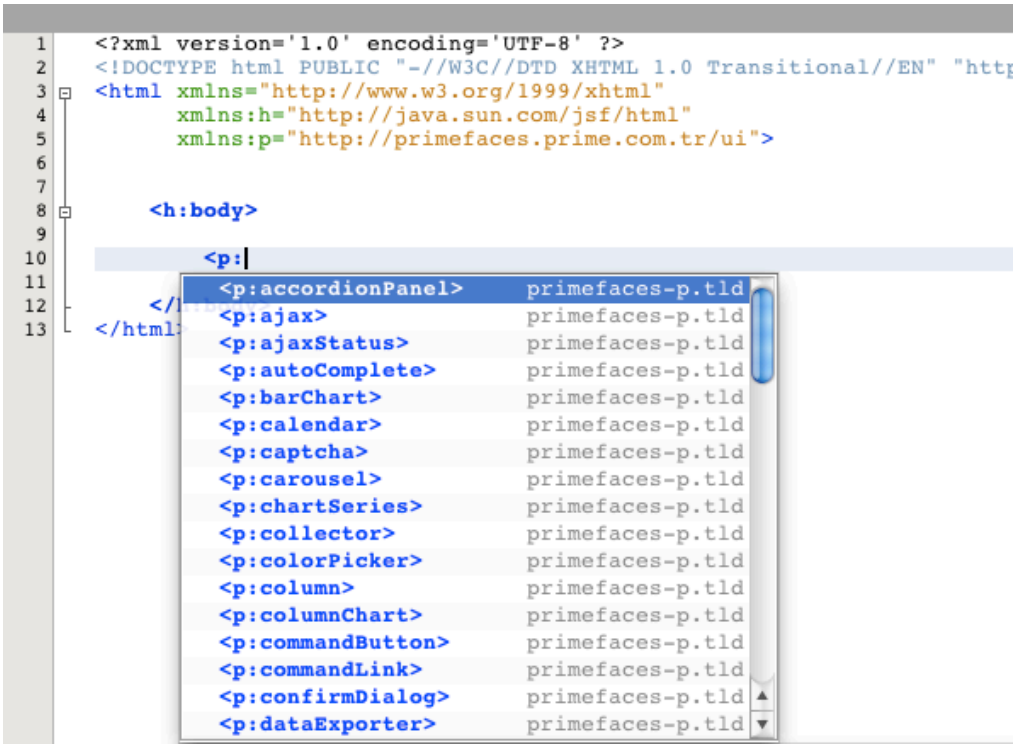
12. IDE Support

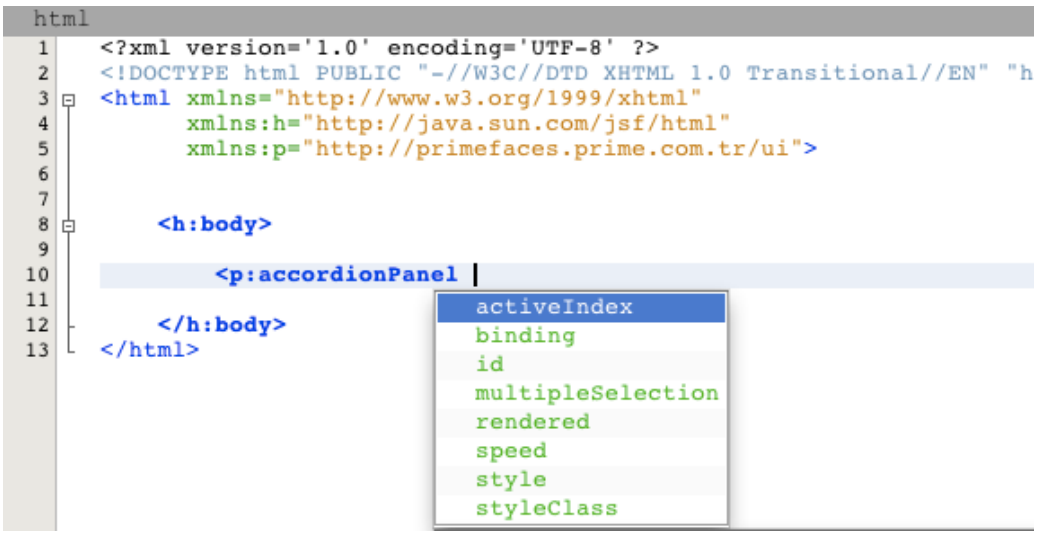
12.1 NetBeans

NetBeans 7.0+ bundles PrimeFaces, when creating a new project you can select PrimeFaces from components tab;



Code completion is supported by NetBeans 6.9+ ;

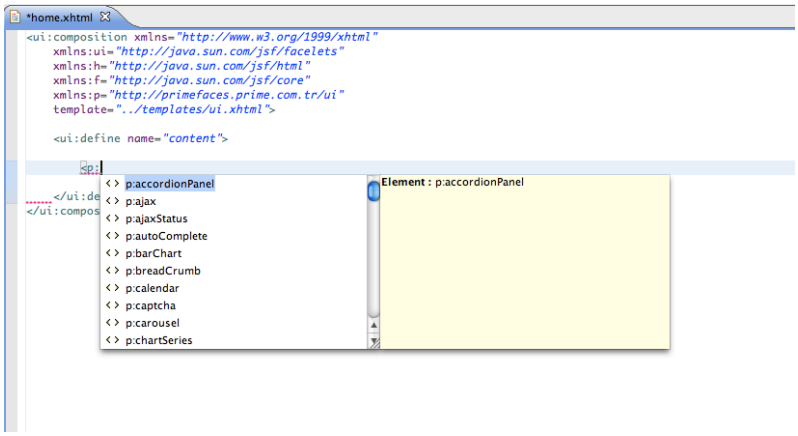


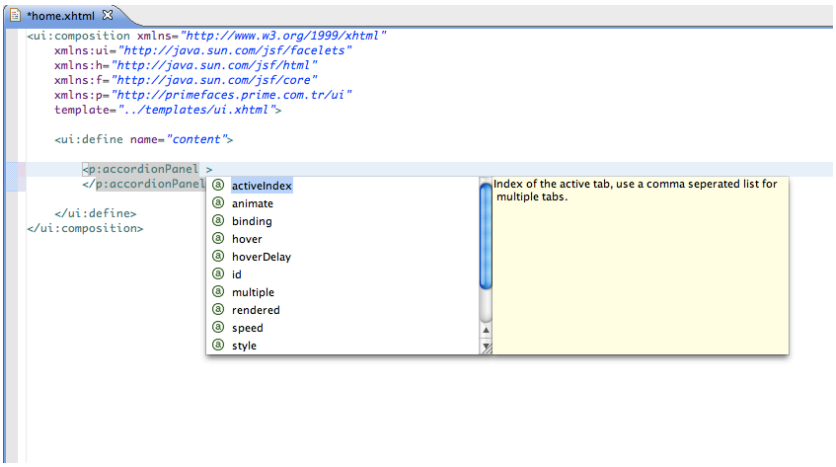


PrimeFaces and NetBeans teams are in communication to discuss the next step of PrimeFaces integration in NetBeans at the time of writing.

12.2 Eclipse

Code completion works out of the box for Eclipse when JSF facet is enabled.





13. Project Resources

Documentation

This guide is the main resource for documentation, for additional documentation like apidocs, taglib docs, wiki and more please visit;

http://www.primefaces.org/documentation.html

Support Forum

PrimeFaces discussions take place at the support forum. Forum is public to everyone and registration is required to do a post.

http://forum.primefaces.org

Source Code

PrimeFaces source is at google code subversion repository.

http://code.google.com/p/primefaces/source/

Issue Tracker

PrimeFaces issue tracker uses google code's issue management system. Please use the forum before creating an issue instead.

http://code.google.com/p/primefaces/issues/list

WIKI

PrimeFaces Wiki is a community driven additional documentation resource.

http://wiki.primefaces.org

Social Networks

You can follow PrimeFaces on twitter using @primefaces and join the Facebook group.

14. FAQ

1. Who develops PrimeFaces?
PrimeFaces is developed and maintained by Prime Teknoloji, a Turkish software development company specialized in Agile Software Development, JSF and Java EE.

2. How can I get support?
Support forum is the main area to ask for help, it's publicly available and free registration is required before posting. Please do not email the developers of PrimeFaces directly and use support forum instead.

3. Is enterprise support available?
Yes, enterprise support is also available. Please visit support page on PrimeFaces website for more information.

http://www.primefaces.org/support.html

4. Where is the source for the example demo applications?
Source code of demo applications are in the svn repository of PrimeFaces at /examples/trunk folder. Snapshot builds of samples are deployed at PrimeFaces Repository time to time.

5. Some components like charts do not work in Safari or Chrome but there's no problem with Firefox.
The common reason is the response mimeType when using with PrimeFaces. You need to make sure responseType is "text/html". You can use the <f:view contentType="text/html"> to enforce this.

6. My page does not navigate with PrimeFaces commandButton and commandLink.?
If you'd like to navigate within an ajax request, use redirect instead of forward or set ajax to false.

7. Where can I get an unreleased snapshot?
Nightly snapshot builds of a future release is deployed at http://repository.primefaces.org.

8. What is the license PrimeFaces have?
PrimeFaces is free to use and licensed under Apache License V2.

9. Can I use PrimeFaces in a commercial software?
Yes, Apache V2 License is a commercial friendly library. PrimeFaces does not bundle any third party software that conflicts with Apache.

10. Which browsers are supported by PrimeFaces?
IE7-8-9, Safari, Firefox, Chrome and Opera.

THE END